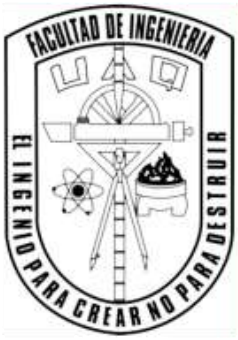
Universidad Autónoma de Querétaro
Facultad de Ingeniería



"Implementación de red multiprotocolo de control y monitoreo de sensores inteligentes"

TESIS

Que como requisito para obtener el título de:

INGENIERO ELECTROMECÁNICO

Presenta:

José Uriel Arroyo Nieto

Dirigido por:

Dr. Luis Morales Velázquez
M. en C. Benigno Muñoz Barrón

Co-Asesor:

Dr. Roque Alfredo Osornio Ríos

San Juan del Río, Qro. abril de 2015.

RESUMEN

En la actualidad el uso de sensores en diversas aplicaciones es un tema que ha crecido y tomado fuerza en gran medida, permitiendo observar que esta tendencia continuará llevándose a cabo en los siguientes años. El amplio uso de sensores se debe en general a las grandes necesidades que se tienen por el conocer la magnitud de alguna variable física de interés, la cual se puede relacionar de manera directa con procesos ya sea de control, monitoreo o ambos. Existe una gran diversidad de sensores que permiten cuantificar variables físicas, pero también existen algunos otros que no solo cuantifican sino que además son capaces de agregar algunas características más a la acción de sensar variables, tales sensores son denominados *sensores inteligentes* y hacen uso de la tecnología actual para proporcionar mejores características a su implementación, ya que aumentan el rendimiento y efectividad del sistema al que pertenecen, al excluir al usuario de parte de carga computacional. Para lograr cierta correspondencia al gran rendimiento que presentan los sensores inteligentes es necesario hacer uso de protocolos y medios de comunicación rápidos, seguros y eficientes que permitan la transferencia de información hasta el punto donde esta pueda ser visualizada o utilizada para la toma de acciones de control. A partir de lo anterior el desarrollo del presente trabajo se lleva a cabo con la finalidad de desarrollar software que permita cubrir los aspectos de comunicación entre un usuario de sistema SCADA y una red conformada por sensores inteligentes y otros dispositivos controladores, donde para tal labor se hace uso de protocolos de comunicación como lo son el protocolo en desarrollo Enhanced ShockBurst™, el protocolo Modbus RTU, así como también del protocolo Ethernet como medio principal de comunicación entre el usuario y la red de trabajo, proporcionando con ello soluciones alternativas de comunicación para procesos industriales de control y monitoreo.

(**Palabras clave**: variable física, sensores inteligentes, sistema, usuario, red, protocolo, Ethernet, Modbus RTU, Enhanced ShockBurst™)

DEDICATORIAS

“A la memoria de mi padre Basilio, a mi madre Cristina

y a mis hermanos Misael, Sergio y Basilio”

AGRADECIMIENTOS

A todas esas personas y amigos que a lo largo de mi desarrollo personal han sido bondadosas y han brindado parte de su tiempo para escucharme, apoyarme y motivarme a seguir adelante y ser una mejor persona.

A mi familia por ser mi inspiración y motivarme.

Al Dr. Luis Morales, al M. en C. Benigno Muñoz y al Dr. Roque, por su apoyo y dirección en la realización de este sencillo pero significativo trabajo.

A todos mis profesores por su esfuerzo y empeño en la formación de personas no solo profesionales sino también con calidad humana.

A la Universidad Autónoma de Querétaro por abrirme sus puertas y formarme profesionalmente.

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Actualmente la implementación de sensores en diversas máquinas y dispositivos ha ido en aumento ya que su uso no se ha restringido simplemente al sector industrial, sino que con los avances logrados en gran variedad de las ramas de la tecnología los sensores han encontrado aplicación en una gran cantidad de aparatos de uso cotidiano para las personas, como lo es en: electrodomésticos, automóviles, aparatos de oficina y en dispositivos relacionados a las telecomunicaciones.

La rapidez y la eficacia en la transferencia de información obtenida mediante el uso de sensores en procesos de control y monitoreo a nivel industrial e investigación se ve limitada por el medio y protocolo de comunicación del cual se hace uso, ya que debido a las características propias de cada uno de ellos y de igual manera a las condiciones del medio ambiente en el cual la comunicación es llevada a cabo se permite al usuario tener un mayor o menor desempeño del sistema.

La integración del medio y protocolo de comunicación según las condiciones y necesidades del sistema, es un tema al cual se debe prestar significativa importancia debido a que es un aspecto fundamental para que la comunicación se lleve a cabo de la mejor manera. Una alternativa que busca reunir las mejores condiciones es él conjuntar varios protocolos que promuevan la interacción de diferentes dispositivos de tal forma que se permita crear una red lo suficientemente flexible para que diversos sensores sean capaces de transmitir información de la variable física que miden hasta un punto donde esta pueda ser utilizada, logrando así con ello una mayor eficiencia en la transferencia de información y en el tiempo utilizado, además de también poder optimizar la energía y los recursos presentes en la red.

Con fines de seguimiento, la estructura de este trabajo se lleva a cabo de la siguiente manera:

En el ***capítulo 1*** se presenta una breve introducción al tema de desarrollo mediante la recopilación de antecedentes relacionados al uso de distintos protocolos de comunicación, para así proporcionar información sobre el desarrollo y resultados obtenidos en trabajos afines al presente. De igual forma también se plantean los objetivos y el planteamiento general del proyecto. Para el ***capítulo 2*** se proporciona la información teórica necesaria para respaldar el desarrollo de este trabajo, donde dicha recopilación se conforma principalmente de temas específicos que involucran de manera directa a los protocolos Ethernet, Modbus y Enhanced ShockBurst™, así como de los modelos de referencia establecidos para la interconexión de dispositivos. En el ***capítulo 3*** se describen paso a paso las etapas realizadas en el desarrollo del proyecto, mostrando desde la manipulación de hardware, hasta la creación y uso de software. Para el ***capítulo 4*** se muestran los resultados obtenidos al llevar a cabo las pruebas de comunicación de los distintos protocolos utilizados, así como los respectivos de la implementación de un sistema SCADA. El ***capítulo 5*** muestra las conclusiones obtenidas a partir del desarrollo completo del proyecto, además de proporcionar cierta prospectiva hacia trabajos futuros. En el ***capítulo 6 y 7*** se elabora un compendio de información relacionada a la integración completa del proyecto, desde la presentación de librerías de software desarrolladas hasta la presentación de códigos fuente para la programación de dispositivos, sin olvidar la debida presentación de las referencias bibliográficas que fueron utilizadas.

1.1 ANTECEDENTES

Al hablar de redes de comunicación entre dispositivos electrónicos y en específico aquellas relacionadas a protocolos de comunicación robustos y flexibles como lo es el protocolo Ethernet, se puede mencionar que años atrás el uso de

este protocolo se restringía solo para la comunicación entre computadoras debido a la complejidad que presenta el medio de enlace a la red y el cual solo poseían estos equipos, pero en la actualidad gracias a los avances tecnológicos y al surgimiento de los llamados controladores embebidos, los cuales han sido definidos por Axelson (1998), como "*computadoras dedicadas a desempeñar simples tareas o un conjunto de tareas relacionadas",* su aplicación ha tomado diferentes rubros ya que ahora es posible la integración de diversos dispositivos controladores de red Ethernet para poder llevar a cabo la implementación de una red bajo este protocolo, la cual puede no solo estar conformada por computadoras sino que ahora a dicha red se pueden integrar como usuarios a diversos dispositivos, como lo son: microcontroladores, controladores lógicos programables (PLC, Programmable Logic Controller), cámaras termográficas y de visión artificial, y en gran medida se intenta conectar algunos otros como los dispositivos electrónicos de alta velocidad FPGA (Field Programmable Gate Array, Arreglo de Compuertas Programables en Campo), potencializando de esta forma el uso de este protocolo en aplicaciones con fines diversos.

De igual forma que lo sucedido con el protocolo Ethernet, el protocolo Modbus y la comunicación inalámbrica, se han encontrado en constante desarrollo ya que en parte al grado de complejidad que presentan los estándares de comunicación y sobre todo los medios físicos, se ha orillado a los usuarios a buscar nuevas alternativas de comunicación comercial que permitan cierto grado de confiabilidad para su aplicación, razón por la cual se pueden encontrar desarrollados controladores embebidos y circuitos integrados de un costo relativamente bajo para estos tipos de comunicación, donde los mismos han sido ya puestos a prueba en distintas aplicaciones y en consecuencia han mostrado un desempeño aceptable.

Desde cualquier punto de vista la gran necesidad de comunicar dispositivos bajo diferentes estándares, así como bajo diferentes medios físicos es un problema con el cual se ha trabajado ya desde hace tiempo a varios niveles

académicos y de investigación, obteniéndose resultados satisfactorios que enmarcan la gran importancia de explotar de mejor manera las características propias de cada protocolo.

Dentro de la Universidad Autónoma de Querétaro se han realizado trabajos en relación al uso e implementación de diversos protocolos de comunicación, los cuales incluyen tanto al protocolo Ethernet, protocolo Modbus RTU, así como a la comunicación inalámbrica por medio de radio frecuencia, agregando que algunos de estos trabajos relacionan de manera directa el uso y aplicación de distintos sensores mediante la implementación de redes de sensores.

Toledano (2006), diseñó e implementó un sistema de monitoreo remoto mediante la integración de comunicación inalámbrica en conjunto con comunicación a través de internet para el monitoreo de variables físicas de importancia presentes en un invernadero, tal como: temperatura y humedad relativa. Realizó la transmisión de datos adquiridos mediante unidades remotas dispuestas dentro de un invernadero a una unidad base, para así con ello mediante un ordenador conectado a través de un puerto USB (Universal Serial Bus, Bus en Serie Universal) generar los gráficos correspondientes del comportamiento del sistema y así acceder a esta información a través de internet, todo esto fue posible gracias a la integración de la tarjeta de comunicación inalámbrica desarrollada UAQ-RF05. El trabajo permitió observar que aunque se posea un medio de comunicación primaria para adquirir los datos al monitorear un sistema, la conexión lograda con internet proporciona un medio para aumentar en gran medida el tamaño de una red, así como para poder llevar a cabo monitoreo de manera remota a largas distancias.

Tres años más adelante Guzmán y Velasco (2009), implementaron el control y monitoreo de un sistema de riego comprendido de 2 zonas en las cuales

instalaron 4 aspersores en cada una de ellas, dicho control comprendió tres opciones de control de riego:

1) activación de control manual.- El usuario controlaba el encendido y apagado de 1 bomba y 2 válvulas.
2) Activación de control automático.- El usuario determinaba los horarios de encendido y apagado del sistema para cada día de la semana y para cada zona de riego, así de esta manera este comportamiento se llevaba de manera automática al correr el sistema.
3) Activación de control semi-automático.- En esta modalidad el usuario determinaba los horarios de encendido del sistema, pero el apagado era realizado por un sensor de humedad de acuerdo a un set-point configurable.

Para la interacción del sistema de control con el usuario se desarrolló una interfaz en LabView® a manera de panel en la cual se podía elegir el modo de control. El sistema se llevó a cabo conectando un PLC a una red Ethernet mediante un módulo especial en conjunto con el desarrollo en el software Rslogix de un programa en diagrama escalera. El análisis del proyecto permitió observar que al implementarse la red para un solo protocolo de comunicación se tiene cierto grado de inflexibilidad para aceptar la conexión de dispositivos o sensores comunicados bajo un distinto protocolo.

Duarte (2009), desarrolló un sistema de adquisición de datos y de control de dispositivos de encendido/apagado aplicado a la acuicultura, donde tal sistema podía ser supervisado y administrado de manera remota mediante técnicas de control comercial en complemento con control difuso. Logró implementar una estación de control que se comunicaba vía inalámbrica con una o varias estaciones de campo, formando de esta manera una red de comunicación con el fin de medir variables de interés en el agua tal como: temperatura, pH, oxígeno

disuelto, nivel y turbidez, y así con ello satisfacer las necesidades de control y monitoreo en granjas acuícolas. La solución llevada a cabo en este proyecto mostró de igual manera la desventaja de aplicación específica de un solo protocolo de comunicación, difiriendo que en este caso se trabajó con comunicación inalámbrica.

De igual forma Vera (2010), generó un sistema inalámbrico para adquirir, procesar y analizar las señales de las vibraciones presentes en una máquina de montaje de componentes de chip, lo cual llevó a cabo mediante el uso de sensores de tecnología MEMS (Micro Electro Mechanical System, Sistema Micro-Electromecánico), un dispositivo FPGA y un protocolo de comunicación estándar por radiofrecuencia como módulo de coordinación de sensores inalámbricos. Para tal acción diseñó el módulo de adquisición e instrumentó un transductor de aceleración, y como un nodo de la red de sensores implementó en hardware los circuitos necesarios para llevar a cabo la comunicación entre la computadora y el módulo coordinador, todo con el propósito de incorporar dicho sistema al monitoreo de las vibraciones de la máquina y así conocer la dinámica del sistema en condiciones normales de operación. Su desarrollo mostró que el uso de sensores inteligentes permite medir eficientemente variables físicas y que como tal proporciona mejores características al monitoreo de variables.

Poco tiempo después Curiel (2011), diseño e implementó un sistema de adquisición de datos para las variables de humedad y temperatura, haciendo uso de una red inalámbrica de sensores (WSN, Wireless Sensor Network) en complemento con un modelo predictivo desarrollado mediante el uso de redes neuronales artificiales, todo realizado con la finalidad llevar a cabo el monitoreo del comportamiento térmico de distintos tipos de recubrimientos de techo y así predecir de igual manera el comportamiento térmico dentro del inmueble, para la comunicación inalámbrica de los sensores se hizo uso del protocolo Zigbee.

González (2012), diseño e implementó un sistema de control de iluminación inteligente en interiores de edificios haciendo uso del monitoreo de una red de sensores lumínicos, donde el controlador del sistema fue basado en lógica difusa. Para el monitoreo de los sensores se hizo uso de módulos de comunicación inalámbrica Xbee del fabricante MaxStream, los cuales funcionan bajo el protocolo de comunicación estándar Zigbee. El proyecto se desarrolló para presentar alternativas que aseguren el confort visual de los usuarios de dichos sistemas, así como para de igual forma permitir reducir el consumo de energía eléctrica al optimizar el uso de la iluminación natural.

No hace mucho tiempo Lugo (2013), implementó una red Ethernet conformada por un ordenador, una cámara termográfica y un microcontrolador que fungía como etapa intermedia para acceder a los vastos recursos de un dispositivo FPGA y así poder llevar a cabo procesamiento de imágenes. Este proyecto se desarrolló con la finalidad de ofrecer una solución alternativa al problema de la velocidad de procesamiento de imágenes aplicado al análisis termográfico, para lo cual se conjunto un dispositivo FPGA por su capacidad y su alta velocidad de procesamiento, y para no contrastar la rápida velocidad de procesamiento adquirida se adicionó como medio de enlace una red de comunicación industrial Ethernet con el fin de garantizar una plataforma de intercambio de datos confiable, estable y veloz.

Meses después Mejía (2013), desarrolló monitoreo remoto vía Ethernet a un sistema basado en FPGA llevándolo a cabo por medio de un microcontrolador que era el encargado de controlar no solo la tarjeta controladora de red, sino que además adquiría los datos provenientes del FPGA y también sobre el mismo dispositivo se encontraba embebido un sitio web al cual era posible acceder desde cualquier punto donde se contará con conexión a internet o a una conexión de área local. Todo esto se desarrolló con el propósito de monitorear de manera remota y periódica los principales parámetros de la calidad de la energía en sistemas de corriente continua y en sistemas monofásicos de corriente alterna.

Fuera de esta casa de estudios y del estado de Querétaro se han desarrollado trabajos afines entorno a la implementación de diversos protocolos de comunicación. Durán y Gutiérrez (2009), diseñaron y llevaron a cabo un sistema capaz de trasmitir y adquirir datos a través de una red Ethernet de manera remota complementada con la creación de un servidor que permitía visualizar la información obtenida al aplicar monitoreo mediante cámaras de video.

El tema concerniente a la comunicación entre dispositivos también ha sido objeto de interés a nivel internacional. Iberico (2010), diseñó y simuló un sistema de seguridad para un edificio de departamentos basándose en una WLAN (Wireless Local Area Network, Red de Área Local Inalámbrica) de tecnología WI-FI y el protocolo de comunicación inalámbrica Zigbee. Integró un sistema de red en el que conjuntó sensores de movimiento, detectores de humo, alarmas contra robo e incendios, control de accesos y manejo de iluminación con el propósito de ofrecer alternativas en cuestiones generales de seguridad en departamentos.

Gordon y Vásquez (2012), realizaron un estudio para permitir conocer la funcionalidad y desempeño en laboratorio de redes inalámbricas de sensores y así sustentar la posibilidad de su aplicación a la monitorización volcánica en tiempo real, todo esto mediante el modelado y la simulación de una red de sensores en el entorno del software de simulación ns-2 (Network Simulator 2) y para su aplicación real haciendo uso del esquipo SmartMesh Industrial-Evaluation Kit del fabricante Dust Networks. El análisis llevado a cabo presentó las ventajas y desventajas del uso de redes inalámbricas de sensores, mostrando que el mal desempeño de una red de esta naturaleza se debe en gran medida a las características del medio ambiente de aplicación, entre otros, permitiendo observar que para resolver estas problemáticas se deben agregar mejores características a la red para así disminuir su vulnerabilidad.

De acuerdo a lo analizado a lo largo de los proyectos revisados, se aprecia que se tiene la necesidad de encontrar un medio que encare principalmente los siguientes problemas en las redes de comunicación:

- Poca flexibilidad de adaptación, hacia sensores y dispositivos controladores comunicados bajo distintos protocolos de comunicación.
- Incompatibilidad de medios físicos de comunicación.
- Incompatibilidad de las tramas de datos (formato del paquete) manejadas por cada protocolo.
- Diferencia de tasas de transferencia entre protocolos.

1.2 DESCRIPCIÓN DEL PROBLEMA

Al abordar el tema de control y monitoreo mediante redes de sensores implementadas bajo distintos protocolos, los proyecto presentados como antecedentes son de gran ayuda para detectar las principales problemáticas de la implementación de redes para cada protocolo en particular, pero también enmarcan que el uso de redes de sensores es una buena alterativa para llevar acabo de una mejor manera el monitoreo de sistemas.

Al analizar detenidamente el funcionamiento de los distintas redes se puede concluir que su metodología para la solución del problema en cada uno de ellos se centra solo en algunos aspectos en específico, es decir que algunos sistemas de acuerdo a su estructura buscan una transferencia de información donde la confiabilidad y la rapidez tiene prioridad en el proceso (Lugo, 2013), mientras que algunos otros buscan optimizar la energía, tener poca perdida de información y que la red sea viable, como lo es en el caso de las redes inalámbricas y para lo cual es de observarse que es difícil reunir tales características en un solo sistema.

Los proyectos desarrollados presentan una solución a cada problema en particular, pero en ninguno se trata de obtener un balance entre las características y propiedades de la red de comunicación establecida de una manera general, es decir que no se busca agregar al sistema mejores características, todos éstos se restringen a las ventajas y desventajas del protocolo de comunicación del cual se hizo uso. Esto permite observar uno de los principales problemas de la implementación de redes, el cual es la poca flexibilidad que la red puede tener para aceptar cambios en su estructura ya que en algunos casos por ejemplo, la comunicación inalámbrica podría ser más conveniente que la comunicación cableada y para lo cual si se tiene una red implementada de manera cableada no existe un medio para integrar a esta red módulos de sensores o controladores implementados bajo comunicación inalámbrica y lo mismo de manera contraria.

Es de destacarse que tal vez una de las razones por el cual estas redes propuestas se diseñaron e implementaron de manera particular para un solo protocolo como medio principal, fue debido a que no solo el medio físico de transmisión de información (cable o aire) puede diferir entre redes sino que además el formato, las tasas de transferencia, y el tamaño de los paquetes de datos no son compatibles, con lo cual se agrega mayor dificultad a la posible conexión de redes implementadas bajo diferentes protocolos.

A todo lo anterior se puede agregar que de manera comercial no se cuenta con un sistema que logre la compatibilidad directa entre diversos protocolos de comunicación utilizados por sensores y controladores genéricos, y que debido a esto uno de los problemas que se presentan en la actualidad es el que en algunos casos si se requiere monitorear un sistema mediante varios sensores comunicados bajo distintos protocolos, o no necesariamente distintos, es necesario hacer uso de un ordenador de manera independiente para cada uno de ellos, lo cual agrega mayor dificultad a la implementación de sistemas de monitoreo.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Desarrollar e implementar una red multiprotocolo de comunicación entre dispositivos, mediante la integración de los protocolos: Ethernet, Enhanced ShockBurst™ y Modbus RTU, para el control y monitoreo de procesos.

1.3.2 OBJETIVOS PARTICULARES

I).- Desarrollar las librerías de software necesarias para el uso y control a través del *sistema embebido en FPGA*, de la tarjeta controladora de red Ethernet de desarrollo ENC28J60-H y así poder transferir datos a nivel IP dentro de una red de área local.

II).- Realizar y analizar la transferencia de información mediante protocolo Modbus RTU, la comunicación por radiofrecuencia y el protocolo Ethernet de manera particular para cada protocolo, de las mediciones realizadas mediante sensores inteligentes.

III).- Realizar y analizar la transferencia de información lograda a partir de la combinación del protocolo Ethernet, la comunicación por radiofrecuencia y la comunicación serial, de las mediciones realizadas mediante sensores inteligentes.

IV).- Diseñar e implementar una red de sensores inteligentes, dispositivos controladores y controlados, sobre un sistema físico.

1.4 JUSTIFICACIÓN

El proyecto tiene como objetivo el implementar una red de comunicaciones que se adapte a las características de los denominados sistemas SCADA (Supervisory Control And Data Acquisition, Control con Supervisión y Adquisición

de Datos) para tal implementación se crea un sistema de comunicaciones conformado a partir del protocolo Ethernet, el protocolo Enhanced ShockBurst™ (comunicación inalámbrica) y el protocolo Modbus RTU basado en la comunicación serial RS-485, con la finalidad de obtener una red que sea lo suficientemente robusta para asegurar la transmisión de señales de control encendido-apagado (*on/off*), sea capaz de presentar características de flexibilidad de adaptación, trasferencia de información rápida y segura, y que además pueda ser utilizada y modificada de acuerdo a las condiciones y necesidades del medio donde se haga uso de ella.

Para su desarrollo se hace uso de tecnología FPGA, donde de manera específica se trabaja con un sistema de procesamiento embebido en FPGA encargado, en conjunto con algunos otros dispositivos electrónicos, de lograr la compatibilidad de protocolos en cuanto a tasas de transferencia, medios físicos y tramas de datos.

Al implementarse la red de comunicaciones en base a dispositivos FPGA se le permite al usuario poder acceder a varias herramientas digitales que microcontroladores comerciales no integran, ya que tales dispositivos ofrecen una gama de herramientas en hardware para el desarrollo de aplicaciones en periodos muy cortos, además de ser dispositivos muy útiles para el procesamiento de información y señales digitales a muy altas velocidades.

De igual manera se debe tener presente que el monitoreo mediante diversos sensores y en distintos entornos, es algo cotidiano y cada vez más frecuente ya que aplicaciones científicas, tecnológicas, en medicina, en medio ambiente, etc., buscan acceder a diferentes dispositivos para llevar a cabo el monitoreo de variables de importancia, razón por la cual la implementación de una red multiprotocolo representa una propuesta de solución a las necesidades de monitoreo en distintos ámbitos, donde las características más importantes que se pueden otorgar a los usuarios son las de permitir adecuar la red de

comunicaciones dependiendo de los requerimiento del sistema y que además no importando el entorno se puede hacer uso de un protocolo de comunicación estándar de nivel industrial como lo es el protocolo Ethernet.

También gracias a la integración del protocolo Ethernet la implementación de la red se puede llevar a cabo a un bajo costo, ya que como Ethernet debido a su económica implementación y mantenimiento, en conjunto con su capacidad de integración con internet, se sitúa como el principal protocolo del nivel de enlace de comunicación de redes LAN (Local Area Network, Red de Área Local) y así al poseer en diferentes entornos la infraestructura que permite el uso de este protocolo se puede hacer uso de ella para implementar una nueva red sin necesidad de adquirir todos los componentes de nueva cuenta.

Por último, la red multiprotocolo propuesta abate el problema de llevar a cabo monitoreo mediante varios ordenadores, ya que es capaz de concentrar toda la información de un sistema completo en un solo ordenador, es decir que se comporta como una unidad integradora de información. Esto permite lograr implementar redes de sensores y controladores de manera libre y genérica, por ser capaces de soportar varias topologías de red.

1.5 PLANTEAMIENTO GENERAL

Para el desarrollo del proyecto y con el fin de poder implementar la red multiprotocolo es necesaria la conformación de un sistema de integración, tal que mediante interacción software-hardware permita la manipulación de distintos protocolos de comunicación para que diversos sensores, comunicados bajo distintos protocolos puedan transferir la información de la variable física que miden hasta una interfaz de usuario y para que además también se permita él envió de señales de control, tal como se muestra en la Figura 1.1.

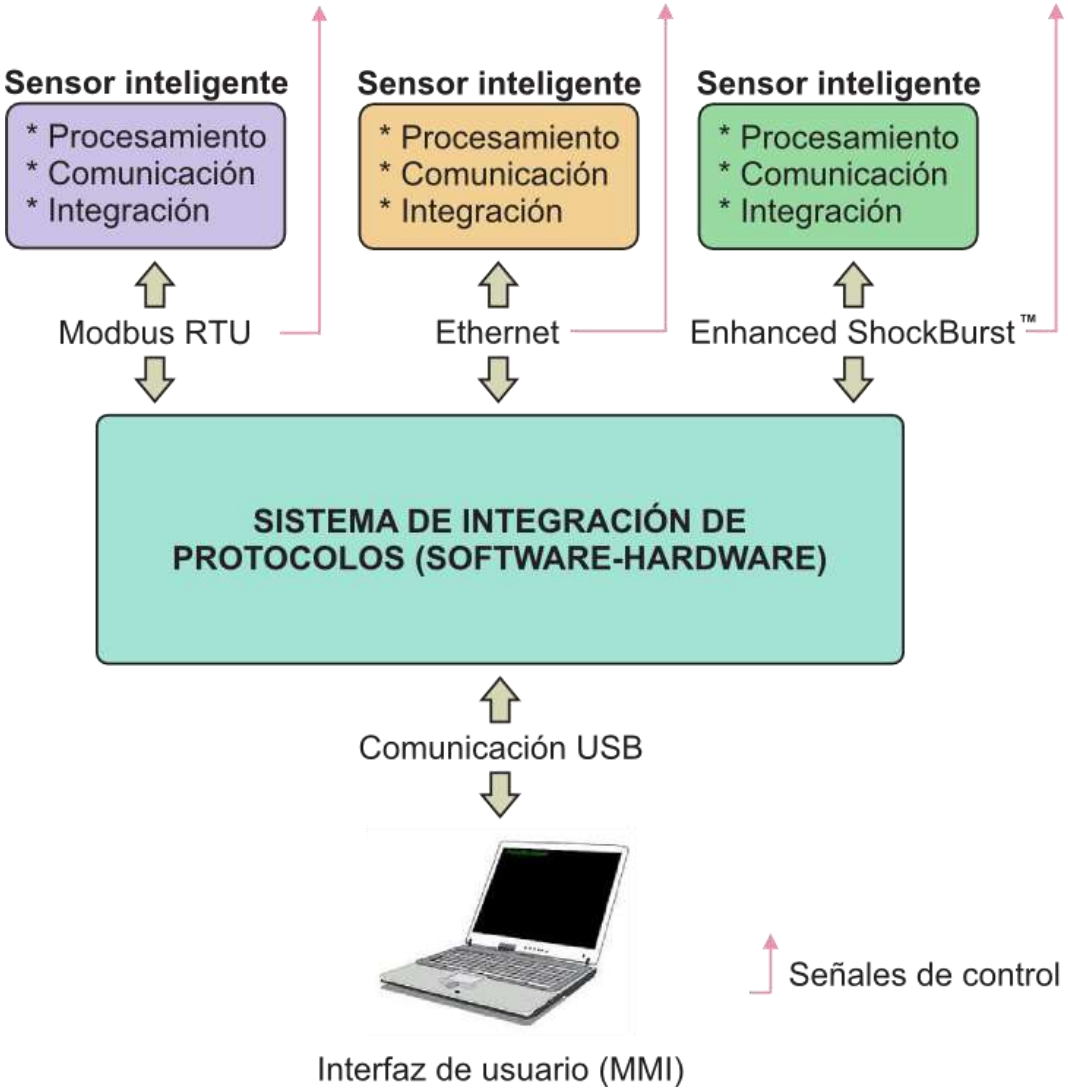


Figura 1.1.- Planteamiento general del proyecto.

El sistema de integración se lleva a cabo mediante el conjunto de diversos dispositivos electrónicos en hardware, así como mediante el uso de herramientas de software y su interacción se aprecia mediante la Figura 1.2.

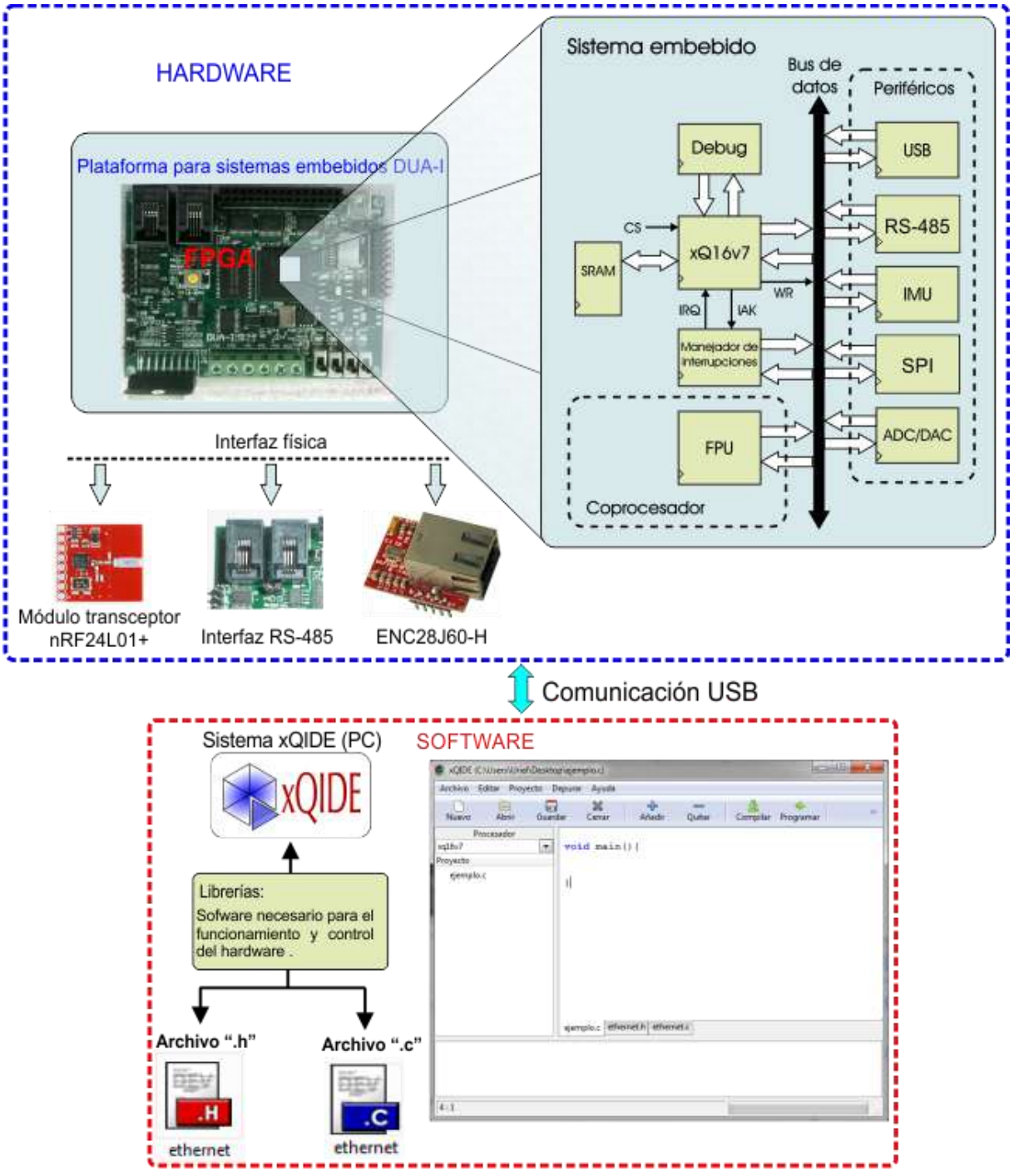


Figura 1.2.- Conformación del sistema de integración.

El hardware se lleva a cabo haciendo uso de la tarjeta electrónica basada en FPGA llamada "*plataforma para sistemas embebidos DUA-I*", la cual alberga dentro del FPGA al "*sistema embebido".* El sistema embebido siendo un sistema implementado mediante descripción de hardware basa su funcionamiento en el procesador xQ16v7, el cual al igual que la tarjeta *DUA-I* es un procesador desarrollado por investigadores de la Universidad Autónoma de Querétaro en la Facultad de Ingeniería campus San Juan del Río.

El xQ16v7 es un procesador de 16 bits pensado como una plataforma para construcción de sistemas embebidos personalizados de alto desempeño y el cual aunque siendo un sistema en desarrollo ya cuenta con varias herramientas tanto en hardware como en software que pueden ser utilizadas para un rápido desarrollo de proyectos.

El sistema embebido además de integrar al procesador xQ16v7 también integra los módulos requeridos para su funcionamiento, tal como un módulo manejador de interrupciones, una unidad de depuración, un módulo de memoria SRAM y se complementa con un coprocesador matemático de punto flotante encargado de realizar las operaciones que involucren números en punto flotante (véase Figura 1.2). La integración de periféricos al sistema consiste en conectar todos los módulos mediante un archivo VHDL que implemente el bus de datos y el ruteo de señales de control (Morales, 2013).

Algunas de las señales de entrada y salida de los módulos periféricos son a su vez mapeados a terminales físicas de la tarjeta *DUA-I.* Para el caso del protocolo Modbus se hace uso del módulo en hardware que implementa la comunicación serial RS-485 y para lo cual su interfaz física se encuentra ya integrada a la tarjeta *DUA-I,* pero para el protocolo Enhanced ShockBurst™ es necesario hacer uso del *módulo transceptor nRF24L01+ con chip antena* del comerciante SparkFun y de igual manera para implementar el protocolo Ethernet es necesaria la *tarjeta controladora de red Ethernet de desarrollo ENC28J60-H*

producida por Olimex®, en ambos casos se hace uso de un periférico que implementa la interfaz SPI (Serial Peripheral Interface, Interfaz Periférica Serie) y que es mapeado al puerto de expansión de la tarjeta *DUA-I*.

En lo que respecta al software, las herramientas de desarrollo para el manejo del procesador xQ16v7 se encuentran integradas en un sistema llamado xQIDE que integra, el compilador C, el ensamblador, el programador y el depurador. El entorno xQIDE además de poseer una interfaz gráfica simple es muy intuitivo, donde solo se requiere crear un nuevo archivo para editar el código del programa, guardarlo y añadirlo al proyecto para proceder a la compilación, de esta forma generando los archivos necesarios para la programación del procesador ya que solo es capaz de ejecutar código de máquina compilado y ensamblado.

El desarrollo de sistemas embebidos personalizados involucra el desarrollo de librerías de software que permitan dar acceso a los módulos en hardware desarrollados. En el caso tanto del módulo que implementa la comunicación serial RS-485, así como para el módulo SPI con el cual se controla la comunicación inalámbrica ya se cuenta con software para su control, pero en cambio para el módulo con el cual se controla la comunicación a través de Ethernet el software necesita ser desarrollado.

En este punto se puede observar claramente que el sistema de integración en cuanto a hardware se refiere se encuentra completo, pero en cambio es necesario desarrollar software para lograr comunicar el sistema embebido a través de Ethernet y para lograr la compatibilidad entre los distintos protocolos.

2. FUNDAMENTACIÓN TEÓRICA

2.1 ESTADO DEL ARTE

En la actualidad las redes de comunicación han crecido a pasos agigantados gracias a los grandes desarrollos tecnológicos que se han obtenido a través del tiempo, pero cabe recordar que su surgimiento fue debido a la gran necesidad del ser humano por comunicarse con sus semejantes y por la de poder transferir información hacia y desde varios puntos, a través de distintas maneras.

Para hablar de redes de comunicación, en cuanto al sistema de red Ethernet, Márquez *et al.,* (2001), mencionan que cuando la red experimental de conmutación de paquetes ARPANET llevaba sólo unos pocos meses de haber entrado en funcionamiento en 1970, un equipo de la Universidad de Hawaii dirigido por Norman Abramson comenzó a experimentar para crear una red que interconectara terminales ubicadas en distintas islas con una computadora central ubicada en una isla ajena a las anteriores. Abramson con su equipo pusieron en marcha una red de radio enlaces, aunque solo asignaron dos canales de frecuencia en dicha red, uno a 413.475 MHz para transmitir del computador central a las distintas islas y el otro a 407.350 MHz para el proceso inverso. El canal de la fase de transmisión no presentaba problemas pues sólo tenía un emisor, sin embargo el canal de retorno era compartido por tres islas por lo cual se hizo necesario establecer reglas que permitieran resolver el problema que se producía cuando varios emisores querían transmitir al mismo tiempo, es decir posibles colisiones, así que se percataron que se requería un protocolo MAC (Media Access Control, Control de Acceso al Medio). La red inventada por este grupo fue llamada ALOHANET y el protocolo MAC utilizado se llamó ALOHA.

En el mismo año de 1970, Robert Matcalfe de igual forma experimentaba con la recién creada ARPANET y conectaba entre sí ordenadores. Matcalfe comenzó a estudiar la red de Abramson y desde un punto de vista teórico planteó

mejoras que se podían hacer al protocolo ALOHA para incrementar su rendimiento lo que años más tarde se convertiría en un nuevo protocolo MAC que recibiría la denominación de *Acceso Múltiple Sensible a la Portadora con Detección de Colisiones* (CSMA/CD, Carrier Sense Multiple Access with Collision Detection) con lo cual planteo una de las características esenciales para el nacimiento de las redes Ethernet actuales. Todo lo anterior y la posterior evolución de Ethernet, es resumida mediante la Figura 2.1.

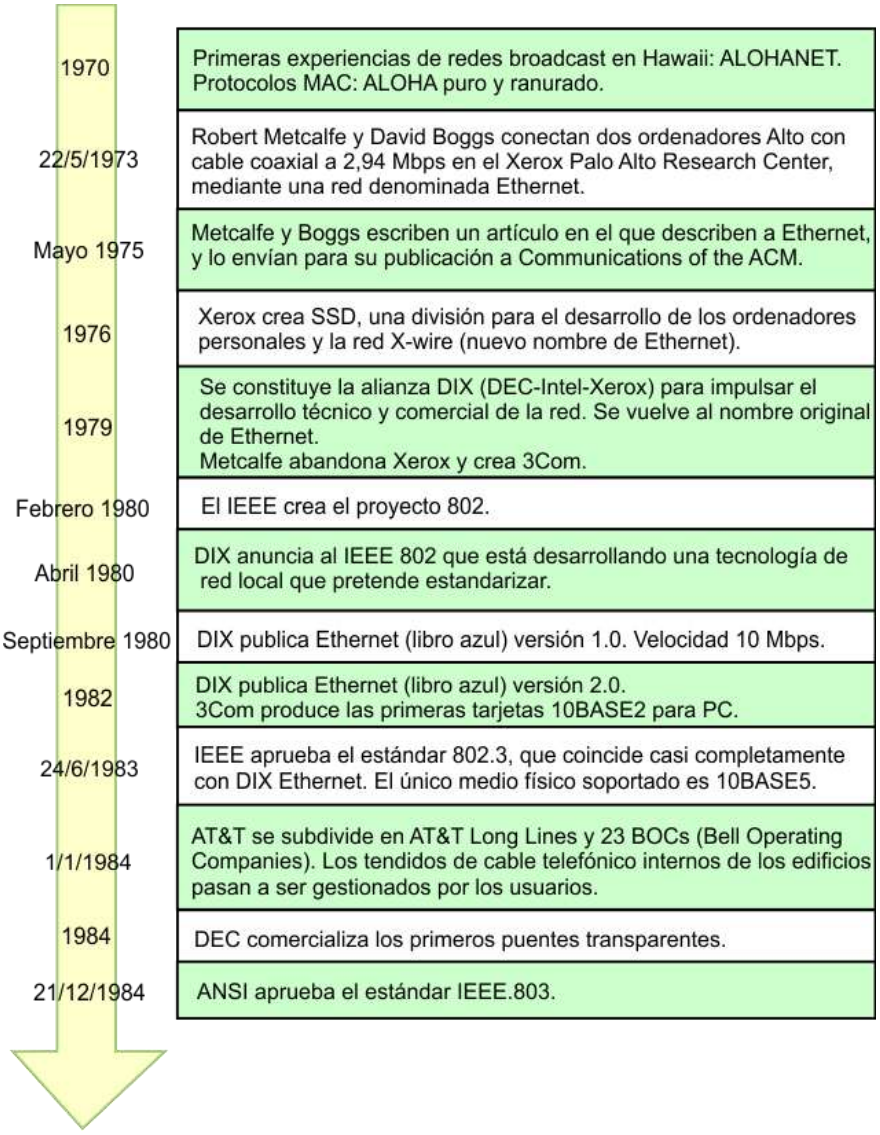


Figura 2.1.- Cronología de Ethernet hasta ser aprobado el estándar. Fuente: Ethernet: Su origen, funcionamiento y rendimiento, Márquez *et al.* (2001).

Para ser aprobado como un estándar de comunicación Stallings (2000), menciona que el comité del IEEE 802.3 planteó las especificaciones necesarias para la transmisión bajo protocolo Ethernet a una velocidad de 10 Mbps, además afirma que de igual forma este comité con el fin de proporcionar una red LAN de bajo coste compatible con Ethernet planteó las especificaciones correspondientes para el Ethernet a alta velocidad (Fast Ethernet, el cual trabaja a 100 Mbps) y para Gigabit Ethernet, donde este último fue diseñado para transmitir paquetes con formato Ethernet a velocidades del orden de Gigabits por segundo.

Para el caso de las redes inalámbricas Mayné (2009), afirma que las primeras redes las implementó IBM en Suiza en el año de 1979 y que este fue el punto de partida de la tecnología WLAN, de igual manera menciona que el crecimiento de este tipo de redes desde su nacimiento hasta los tiempos actuales se debe a que las transmisiones de datos entre equipos electrónicos sin cables se ha aplicado cada vez más debido al desarrollo de nueva tecnología como por ejemplo la creación de circuitos integrados, con lo cual asegura que es posible diseñar redes inalámbricas sin necesidad de tener demasiados conocimientos sobre la transmisión inalámbrica, ni tampoco de disponer de costosa instrumentación para ello.

En cuanto al uso de comunicaciones seriales como enlaces de computadoras, las interfaces RS-232 y RS-485 han ganado popularidad desde su creación hasta la actualidad gracias a que dichas interfaces no solo son capaces de conectar computadoras a una red sino que ahora también pueden conectar controladores embebidos (Axelson, 1998).

2.2 TECNOLOGÍA FPGA: ¿QUÉ ES UN FPGA?

En el nivel más alto, los FPGAs son chips de silicio reprogramables, al utilizar bloques de lógica pre-construidos y recursos para ruteo programables, se

pueden configurar para implementar funcionalidades personalizadas en hardware sólo se debe desarrollar tareas de cómputo digital en software y compilarlas en un archivo de configuración o bitstream que contenga información de cómo deben conectarse los componentes.

La adopción de chips FPGA en la industria ha sido impulsada por el hecho de que combinan lo mejor de los ASICs (Aplication Specific Integrated Circuit, Circuito Integrado de Aplicación Especifica) y de los sistemas basados en procesadores, ofreciendo así velocidades temporizadas por hardware y fiabilidad, pero sin requerir altos volúmenes de recursos. El silicio reprogramable tiene la misma capacidad de ajustarse que un software que se ejecuta en un sistema basado en procesadores, pero no está limitado por el número de núcleos disponibles, a diferencia de los procesadores los FPGAs llevan a cabo diferentes operaciones de manera concurrente y éstas no necesitan competir por los mismos recursos ya que cada tarea de procesos independientes se asigna a una sección dedicada del chip y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica.

2.2.1 LOS CINCO BENEFICIOS DE LA TECNOLOGÍA FPGA

Rendimiento: Los FPGAs exceden la potencia de cómputo de los procesadores digitales de señales (DSP, Digital Signal Processor) rompiendo el paradigma de ejecución secuencial y logrando más en cada ciclo de reloj. El controlar entradas y salidas (E/S) a nivel de hardware ofrece tiempos de respuesta más veloces y funcionalidad especializada que coincide con los requerimientos de diversas aplicaciones.

Tiempo en llegar al mercado: La tecnología FPGA ofrece flexibilidad y capacidades de rápido desarrollo de prototipos para enfrentar los retos de que un producto se libere tarde al mercado. Se puede probar una idea o un concepto y

verificarlo en hardware sin tener que pasar por el largo proceso de fabricación por el que pasa un diseño personalizado de ASIC.

Precio: El precio de la ingeniería no recurrente de un diseño personalizado ASIC excede considerablemente al de las soluciones de hardware basadas en FPGA. La fuerte inversión inicial de los ASICs es fácilmente justificable para los fabricantes de equipos originales que embarcan miles de chips por año, pero muchos usuarios finales necesitan la funcionalidad de un hardware personalizado para decenas o cientos de sistemas en desarrollo con distintos requerimientos que van cambiando con el tiempo.

Fiabilidad: Mientras que las herramientas de software ofrecen un entorno de programación, los circuitos de un FPGA son una implementación segura de la ejecución de un programa. Los sistemas basados en procesadores frecuentemente implican varios niveles de abstracción para auxiliar a programar las tareas y compartir los recursos entre procesos múltiples. El software a nivel driver se encarga de administrar los recursos de hardware y el sistema operativo administra la memoria y el ancho de banda del procesador. El núcleo de un procesador sólo puede ejecutar una instrucción a la vez y estos sistemas están siempre en riesgo de que sus tareas se obstruyan entre sí. Los FPGAs que no necesitan sistemas operativos, minimizan los retos de fiabilidad con ejecución concurrente y hardware preciso dedicado a cada tarea.

Mantenimiento a largo plazo: Los chips FPGA son reconfigurables en campo y no requieren el tiempo y el precio que implica rediseñar un ASIC. Los protocolos de comunicación digital por ejemplo, tienen especificaciones que podrían cambiar con el tiempo y las interfaces basadas en ASICs podrían causar retos de mantenimiento y habilidad de actualización, los chips FPGA al ser reconfigurables son capaces de mantenerse al tanto con modificaciones a futuro que pudieran ser necesarias (National Instruments, 2011).

2.3 SISTEMAS SCADA

Se le da el nombre de SCADA a cualquier software que permita el acceso a datos remotos de un proceso y permita, utilizando las herramientas de comunicación necesarias en cada caso, el control del mismo.

A partir de la definición es posible observar que no se trata de un sistema de control sino de una utilidad software de monitorización o supervisión que realiza la tarea de interfaz entre los niveles de control (actuadores) y los de gestión a un nivel superior.

Los objetivos para que su instalación sea perfectamente aprovechada son los siguientes:

- Funcionalidad completa de manejo y visualización en sistema operativo Windows sobre cualquier PC estándar.
- Arquitectura abierta que permita combinaciones con aplicaciones estándar y de usuario, que permitan a los integradores crear soluciones de mando y supervisión optimizadas (Active X para ampliación de prestaciones, OPC para comunicaciones con terceros, OLE-DB para comunicación con bases de datos, lenguaje estándar integrado como Visual Basic o C, acceso a funciones y datos mediante API).
- Sencillez de instalación sin exigencias de hardware elevadas, fáciles de utilizar y con interfaces amigables con el usuario.
- Fácilmente configurable y escalable, debe ser capaz de crecer o adaptarse según las necesidades cambiantes de una empresa.
- Ser independiente del sector y la tecnología.
- Funciones de mando y supervisión integradas.
- Comunicaciones flexibles para poder comunicar con total facilidad y de forma transparente al usuario con el equipo de planta y con el

resto de la empresa (redes locales y de gestión). La topología de un sistema SCADA variará adecuándose a las características de cada aplicación, unos sistemas funcionarán bien en configuraciones de bus, otros en configuraciones de anillo, etc.

2.3.1 OBJETIVOS DE UN SISTEMA SCADA

Los sistemas SCADA se conciben principalmente como una herramienta de supervisión y mando, entre algunos de sus objetivos se pueden destacar:

- *Economía:* Es más fácil ver qué ocurre en la instalación desde la oficina que enviar a un operario a realizar la tarea. Ciertas revisiones se convertirán innecesarias.
- *Accesibilidad:* Un sistema al completo se puede tener en un clic de ratón encima de la mesa de trabajo.
- *Mantenimiento:* La adquisición de datos materializa la posibilidad de obtener datos de un proceso, almacenarlos y presentarlos de manera inteligible para un usuario no especializado.
- *Gestión:* todos los datos recopilados pueden ser valorados de múltiples maneras mediante herramientas estadísticas, gráficas, valores tabulados, etc., que permitan explotar el sistema con el mejor rendimiento posible.
- *Flexibilidad:* cualquier modificación de alguna de las características del sistema de visualización no significa un gasto en tiempo y medios, pues no hay modificaciones físicas que requieran la instalación de un cableado o del contador.
- *Conectividad:* se buscan sistemas abiertos, es decir sin secretos ni sorpresas para el integrador. La documentación de los protocolos de comunicación actuales permite la interconexión de sistemas de

diferentes proveedores y evita la existencia de lagunas informativas que puedan causar fallos en el funcionamiento o en la seguridad.

El IEEE (Institute of Electrical and Electronic Engineers, Instituto de Ingenieros Eléctricos y Electrónicos) define como sistema abierto todo aquel que proporciona los medios para poder funcionar correctamente con otros sistemas que operen bajo las mismas especificaciones que éste, siendo estas especificaciones de dominio público.

Todos los sistemas de mayor o menor complejidad, orientados a lo anteriormente dicho, aparecen bajo uno de los nombres más habituales para definir esta relación:

MMI: *Man Machine Interface*, Interfaz Hombre-Máquina.

HMI: *Human Machine Interface*, Interfaz Humano-Máquina.

El sistema a controlar aparece ante el usuario bajo un número más o menos elevado de pantallas con mayor o menor información. Se pueden encontrar planos, esquemas eléctricos, gráficos de tendencias, etc., (Rodríguez, 2007).

2.4 TOPOLOGÍAS DE RED

Las topologías LAN básicas y más comunes soportadas por la capa física de varias redes existentes son las de bus, árbol, anillo y estrella.

Topología en bus y en árbol: Se caracterizan por el uso de un medio multipunto. En el caso de la topología en bus, todas las estaciones se encuentran directamente conectadas a través de interfaces físicas apropiadas a un medio de transmisión lineal o bus, es decir que una transmisión desde cualquier estación se propaga a través del medio en ambos sentidos y es recibida por el resto de estaciones (véase Figura 2.2 (a)). La topología en árbol es una generalización de

la topología en bus (véase Figura 2.2 (b)), el medio de transmisión es un bucle ramificado sin bucles cerrados, que comienza en un punto conocido como raíz o cabecera. Uno o más cables comienzan en el punto raíz, y cada uno de ellos puede presentar ramificaciones, a su vez estas ramas pueden disponer de ramas adicionales dando lugar con ello a esquemas más complejos.

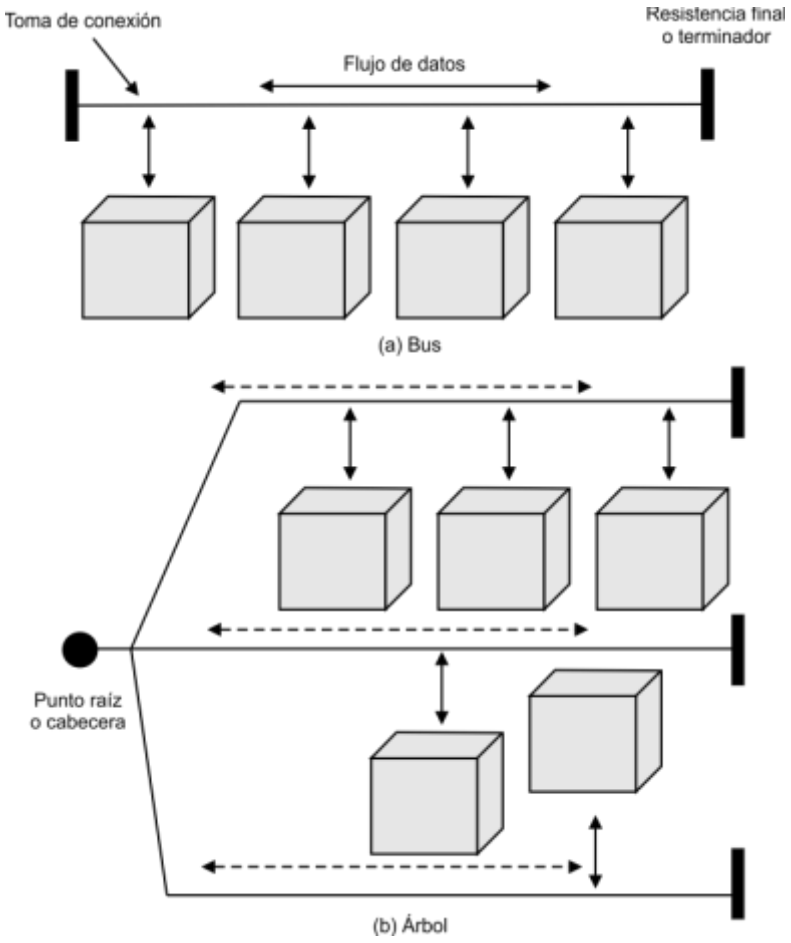


Figura 2.2.- Topologías LAN en bus y en árbol.

Topología en anillo: En esta topología la red consta de un conjunto de repetidores unidos por enlaces punto a punto formando así un bucle cerrado. El repetidor es un dispositivo relativamente simple, capaz de recibir datos a través del enlace y de transmitirlos bit a bit a través del otro enlace tan rápido como son recibidos.

Los enlaces son unidireccionales, es decir que los datos se transmiten sólo en un sentido de modo que éstos circulan alrededor del anillo en el sentido de las agujas del reloj o en el sentido contrario, tal como se muestra en la Figura 2.3.

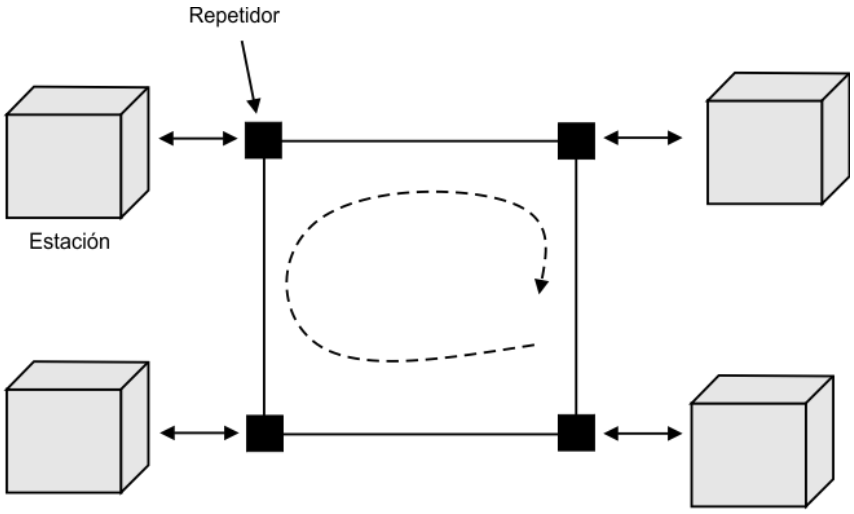


Figura 2.3.- Topología LAN en anillo.

Topología en estrella: En estas redes cada estación está directamente conectada a un nodo central generalmente a través de dos enlaces punto a punto, uno para transmisión y otro para recepción (Figura 2.4).

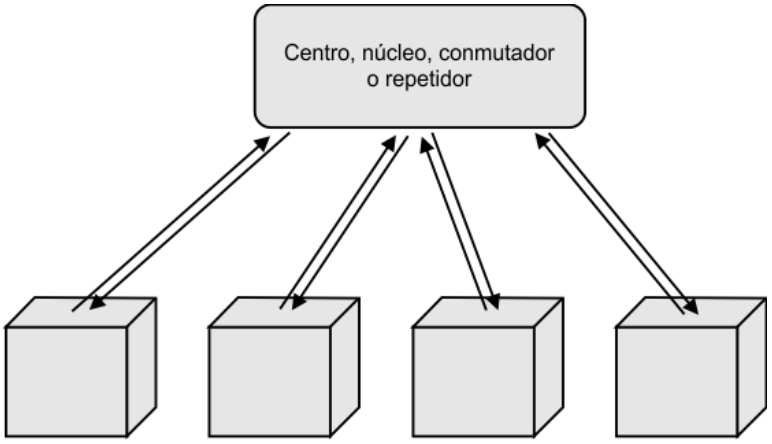


Figura 2.4.- Topología LAN en estrella.

En general existen dos alternativas para el funcionamiento del nodo central. Una es el funcionamiento en modo de difusión, en el que la transmisión de una trama por parte de una estación se transmite sobre todos los enlaces de

salida del nodo central. Otra aproximación es el funcionamiento del nodo central como dispositivo de conmutación de tramas. Una trama entrante se almacena en el nodo y se retransmite sobre un enlace de salida hacia la estación de destino.

2.5 MODELOS Y ARQUITECTURAS DE INTERCONEXIÓN DE SISTEMAS

Existen dos arquitecturas que han sido determinantes y básicas en el desarrollo de los estándares de comunicación: el conjunto de protocolos TCP/IP (Transmission Control Protocol / Internet Protocol) y el modelo de referencia OSI (Open System Interconnection). TCP/IP es la arquitectura más adoptada para la interconexión de sistemas en la actualidad, mientras que OSI se ha convertido en el modelo estándar para clasificar las funciones de comunicación.

2.5.1 LA ARQUITECTURA DE PROTOCOLOS TCP/IP

TCP/IP es el resultado de la investigación y desarrollos llevados a cabo en la red experimental de conmutación de paquetes ARPANET y se denomina globalmente como la familia de protocolos TCP/IP. Esta familia consiste en una extensa colección de protocolos que se han erigido como estándares de Internet.

Al contrario que en el modelo OSI no existe un modelo oficial de referencia TCP/IP, mas sin embargo tomando como base los protocolos estándar que se han desarrollado, todas las tareas involucradas en la comunicación se pueden organizar en cinco capas relativamente independientes, las cuales son:

- Capa física.- Define la interfaz física entre el dispositivo de transmisión de datos y el medio de transmisión o red.
- Capa de acceso a la red.- Es la responsable del intercambio de datos entre el sistema final y la red a la cual se está conectado. El

emisor debe proporcionar a la red la dirección del destino, de tal manera que la red pueda encaminar los datos hasta el destino apropiado.

- Capa internet.- Es la encargada de llevar a cabo los procedimientos necesarios para que los datos atraviesen distintas redes interconectadas en el caso en que dos dispositivos se encuentren conectados en redes diferentes o para encaminar los datos al dispositivo correcto si es que varios dispositivos pertenecen a la misma red.
- Capa origen-destino o de transporte.- En esta capa se encuentran los procedimientos que garantizan la transmisión segura de los datos ya que no importando la naturaleza de las aplicaciones que están intercambiando datos, usualmente es requerido que este intercambio sea seguro.
- Capa de aplicación.- Contiene la lógica necesaria para posibilitar las distintas aplicaciones de usuario, donde cada una de ellas necesita de un módulo bien definido.

La arquitectura TCP/IP puede ser visualizada desde otra perspectiva a partir de la Figura 2.5.

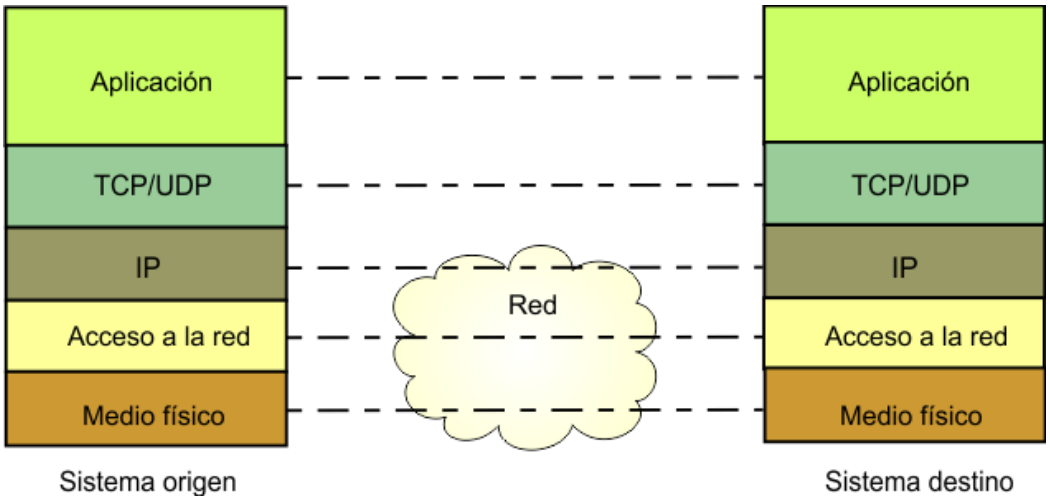


Figura 2.5.- Modelo de capas utilizado por la arquitectura TCP/IP.

2.5.2 MODELO DE REFERENCIA OSI

El modelo OSI se desarrolló por la Organización Internacional de Estandarización (ISO, International for Standarization Organization) como una arquitectura para comunicaciones entre computadoras, con el objetivo de ser referencia en el desarrollo de protocolos estándares, OSI considera siete capas: Aplicación, Presentación, Sesión, Transporte, Red, Enlace de datos y Física.

La intención del modelo OSI es que los protocolos se desarrollen de forma tal que realicen las funciones de cada una de las capas, las cuales se muestran en la Figura 2.6.

Figura 2.6.- Capas de OSI.

2.5.3 PROTOCOLO IP

El Protocolo Internet (IP) es parte del conjunto de protocolos TCP/IP y es el protocolo de interconexión entre redes más utilizado, donde como cualquier otro protocolo estándar, IP se especifica en dos partes:

- La interfaz con la capa superior (TCP o UDP), especificando los servicios que proporciona IP.
- El formato real del protocolo y los mecanismos asociados.

El protocolo entre entidades IP se puede describir mejor tomando como referencia al formato del datagrama IP mostrado en la Figura 2.7, donde sus campos son los siguientes:

- Versión.- Indica el número de la versión del protocolo, para así permitir la evolución del mismo.
- Longitud de la cabecera de internet (IHL, Internet Header Length).- Longitud de la cabecera expresada en palabras de 32 bits.
- Tipo de servicio.- Especifica los parámetros de seguridad, prioridad, retardo y rendimiento.
- Longitud total.- Longitud total del datagrama en octetos.
- Identificador.- Un número de secuencia que, junto a la dirección origen y destino y el protocolo usuario, se utilizan para identificar de forma única un datagrama.
- Indicadores.- Se utilizan para especificar si el datagrama se puede fragmentar y reensamblar.
- Desplazamiento del fragmento.- Indica el lugar donde se sitúa el fragmento dentro del datagrama original, medido en unidades de 64 bits.
- Tiempo de vida.- Especifica cuanto tiempo, en segundos, se le permite a un datagrama permanecer en la red.

- Protocolo.- Usado para especificar el protocolo de la capa superior.
- Suma de comprobación de la cabecera.- Un código de detección de errores aplicado solamente a la cabecera.
- Dirección origen.- Codificada para permitir una asignación variable de bits para poder especificar la red y el sistema final conectado.
- Dirección destino.- Igual que el campo anterior.
- Opciones (variable).-Contiene las opciones solicitadas por el usuario que envía los datos.
- Relleno (variable).-Se usa para asegurarse que la cabecera del datagrama tiene una longitud múltiplo de 32 bits.
- Datos (variable).- Este campo debe tener una longitud múltiplo de 8 bits. La máxima longitud de un datagrama (campo de datos más datagrama) es de 65 533 octetos.

Los campos dirección origen y destino en la cabecera IP contienen cada uno una dirección internet de 32 bits global, que generalmente consta de un identificador de red y un identificador de usuario.

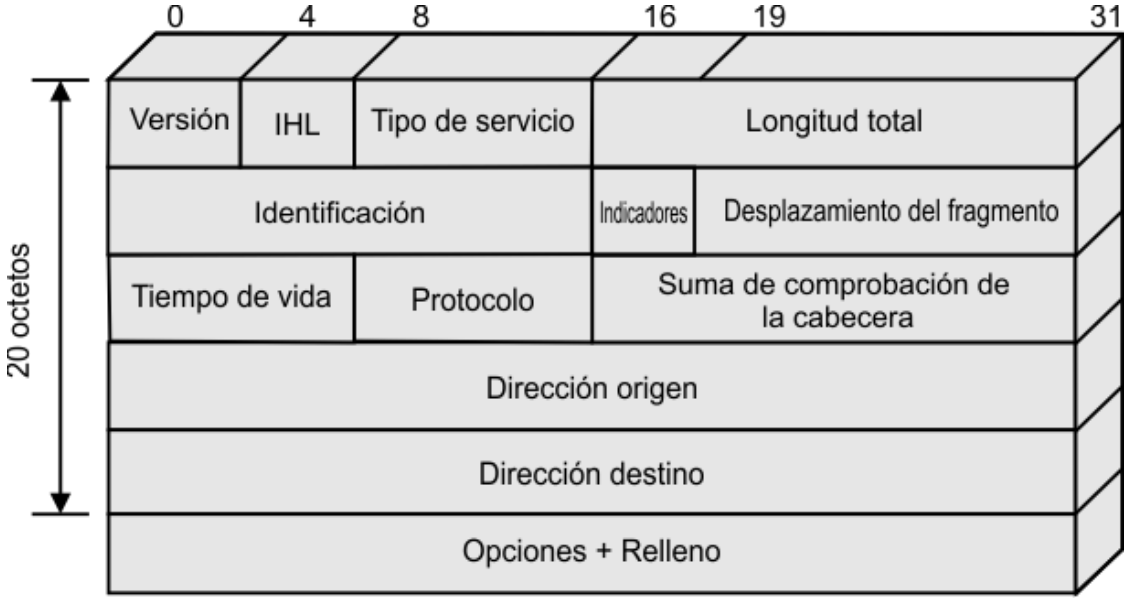


Figura 2.7.- Cabecera IPv4. Fuente: Comunicaciones y redes de computadores, Stallings (2000).

2.5.4 PROTOCOLO ICMP

El estándar IP especifica que una implementación que cumpla las especificaciones del protocolo debe también implementar ICMP (Internet Control Message Protocol, protocolo de mensajes de control de internet). ICMP proporciona un medio para transferir mensajes desde los dispositivos de encaminamiento y otros computadores a un computador. En esencia ICMP proporciona información de retroalimentación sobre problemas entorno a la comunicación.

ICMP maneja varios tipos de mensajes, pero todos ellos empiezan con una cabecera de 64 bits que consta de los siguientes campos:

- Tipo (8 bits).- Específica el tipo de mensaje ICMP.
- Código (8 bits).- Se usa para especificar parámetros del mensaje que se pueden codificar en uno o en unos pocos bits.
- Suma de comprobación (16 bits).- Suma de comprobación del mensaje ICMP entero (cabecera más datos), calculada usando el mismo método que para la suma de la cabecera del protocolo IP.
- Parámetros (32 bits).- Especifica parámetros más largos (Figura 2.8).

Los mensajes ICMP pueden ser del siguiente tipo:

- Destino inalcanzable.
- Tiempo excedido.
- Problemas de parámetros.
- Ralentización del origen.
- Redirección.
- Petición de eco.
- Respuesta a eco.
- Marca de tiempo.
- Respuesta a la marca de tiempo (Stallings, 2000).

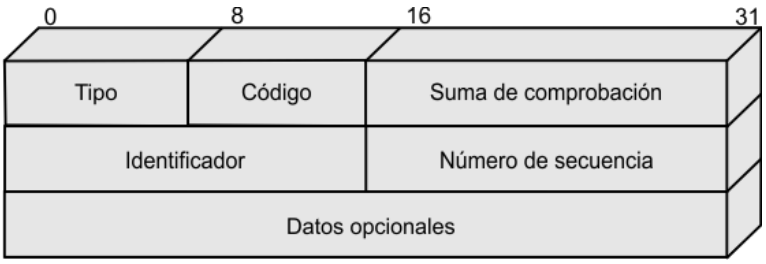


Figura 2.8.- Formato de mensaje ICMP para la petición y respuesta de eco.

2.5.5 PROTOCOLO ARP

La dirección IP de un dispositivo así como también su tabla de encaminamiento permiten enviar un datagrama a una red física específica, pero cuando los datos viajan a través de la red deben obedecer los protocolos usados por el nivel físico. La red física subyacente no entiende las direcciones de red (por ejemplo IP), ya que estas tienen sus propios esquemas de direccionamiento y hay diferentes diseños tanto como redes físicas existen. Una de las tareas del protocolo de acceso a la red es proyectar la dirección de red a la dirección física. El protocolo que logra la traducción de la dirección IP a una dirección MAC es el protocolo ARP (Address Resolution Protocol, Protocolo de Resolución de Direcciones), el cual ha sido definido por la RFC 826.

El protocolo IP brinda una gran transparencia a las aplicaciones de la capa superior: mientras la aplicación conozca su propia dirección IP y la del destino, sabe que puede establecer comunicación. Sin embargo a nivel de capas inferiores (capas de hardware) las direcciones IP no tienen sentido “a nivel de capa física cada host o usuario tiene una dirección de hardware”, la cual depende de la tecnología y los protocolos de la red LAN a la que se encuentre conectado (Ethernet, Token Ring, etc.) y es mediante estas direcciones de hardware que se produce la comunicación real.

La arquitectura de protocolos TCP/IP puede incluir al protocolo ARP (debido a que este protocolo está definido de forma amplia para su uso también en

redes que no estén implementados bajo esta arquitectura). En esencia ARP define dos tipos básicos de mensajes:

- Mensaje de solicitud.
- Mensaje de respuesta.

De manera general se afirma que el mensaje de solicitud incluye la dirección IP del usuario con el cual se desea establecer conexión y solicita la dirección de hardware correspondiente. También esencialmente, se puede decir que un mensaje de repuesta incluye la dirección IP enviada con la solicitud junto con la del hardware asociado. Un mensaje ARP es una larga secuencia de bits, distribuida como se muestra en la Figura 2.9.

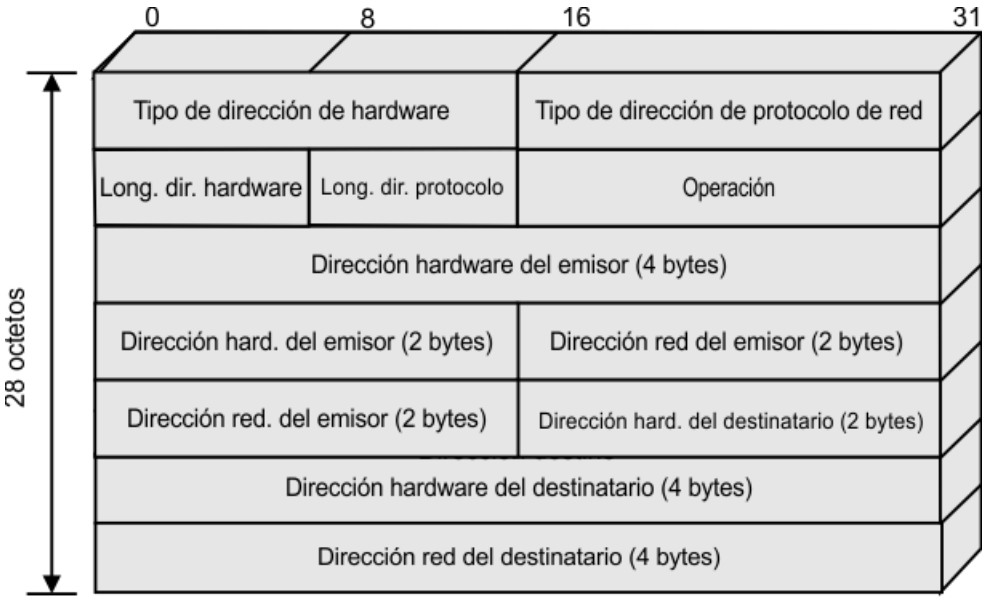


Figura 2.9.- Formato de cabecera del protocolo ARP. Fuente: Comunicaciones y redes de computadores, Stallings (2000).

Los campos del protocolo ARP son los siguientes:

- Tipo de dirección de hardware.- Es un campo de dieciséis bits, indica el tipo de dirección de hardware que es usada.
- Tipo de dirección de protocolo de red.- Indica el tipo de dirección del protocolo de red.

- Longitud de dirección de hardware.- Indica la longitud de la dirección utilizada por el hardware implementado.
- Longitud de dirección de protocolo.- Indica la longitud de la dirección utilizada por el protocolo de red.
- Operación.- De igual forma es un campo de dieciséis bits utilizado para indicar el tipo de operación que se desea realizar, por ejemplo una solicitud o una respuesta.
- Dirección hardware del emisor.- Es un campo de 6 bytes porque en este caso se ha supuesto que el hardware utilizado es Ethernet y es usado para agregar la dirección del hardware comenzando por su byte más significativo.
- Dirección de red del emisor (suponiendo uso de IP).- Es un campo de 4 bytes usado para agregar la dirección de red del emisor (comenzando por el más significativo).
- Dirección hardware del destinatario.- campo utilizado para agregar la dirección de hardware del usuario al que es destinado el mensaje, en un mensaje de solitud, este campo no tiene validez.
- Dirección de red del destinatario (suponiendo uso de IP).- Campo utilizado para agregar la dirección IP del usuario del cual se requiere su dirección física (Cura, 2010).

2.6 ETHERNET (CSMA/CD)

La técnica de control de acceso al medio más ampliamente usada en las topologías en bus y en estrella es la de acceso múltiple sensible a la portadora con detección de colisiones (CSMA/CD). La versión original en banda base de esta técnica fue desarrollada por Xerox para redes LAN Ethernet, este desarrollo fue la base para la posterior especificación del estándar IEEE 802.3 (protocolo Ethernet).

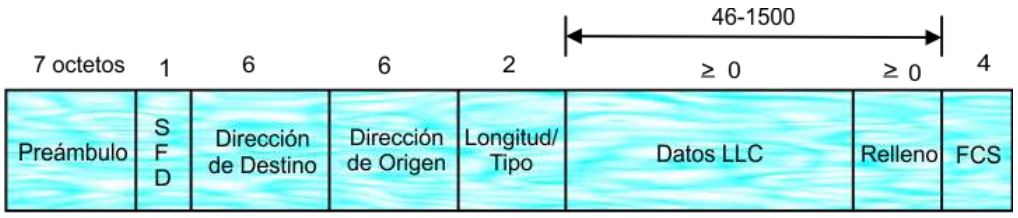
2.6.1 CONTROL DE ACCESO AL MEDIO EN IEEE 802.3

La técnica CSMA/CD y sus precursoras pueden ser denominadas de acceso aleatorio o de contención. Se denominan de acceso aleatorio en el sentido de que no existe un tiempo preestablecido o predecible para que las estaciones transmitan: la transmisión se realiza aleatoriamente. Son de contención en el sentido de que las estaciones compiten para conseguir el acceso al medio.

Las reglas que rigen a CSMA/CD, son:

- 1.- La estación transmite si el medio está libre, si no se aplica la regla 2.
- 2.- Si el medio se encuentra ocupado, la estación continua escuchando hasta que encuentra libre el canal, en cuyo caso transmite inmediatamente.
- 3.- Si se detecta una colisión durante la transmisión, las estaciones transmiten una señal corta de alerta para asegurarse de que todas las estaciones constaten la colisión y cesen de transmitir.
- 4.- Después de transmitir la señal de alerta se espera un intervalo de tiempo de duración aleatoria, tras el cual se intenta transmitir de nuevo (volviendo al paso 1).

Una regla importante aplicada en la mayor parte de los sistemas CSMA/CD, incluyendo a las normalizaciones IEEE, consiste en que la trama debe ser lo suficientemente larga como para permitir la detección de colisiones antes de que finalice la transmisión. Adicionalmente para poder acceder al medio es necesario cierta trama MAC compuesta de varios campos (véase Figura 2.10).



SFD: Delimitación de comienzo de trama (Start of Frame Delimiter).
FCS: Secuencia de comprobación de trama (Frame Check Secuence).

Figura 2.10.- Formato de la trama IEEE 802.3.

Cada campo desempeña:

- Preámbulo.- El receptor utiliza un octeto patrón de 7 bits ceros y unos alternados para establecer la sincronización entre el emisor y el receptor.
- Delimitador del comienzo de la trama (SFD, Start of Frame Delimiter).- Consiste en una secuencia de bits que indica el comienzo real de la trama.
- Dirección de destino (DA, Destination Address).- Especifica la estación o estaciones a las que va dirigida la trama.
- Dirección de origen (SA, Source Address).- Especifica la estación que envió la trama.
- Longitud/Tipo.- Contendrá la longitud del campo de datos LLC expresado en octetos o el campo Tipo Ethernet versión 2 (DIX), dependiendo de que la trama siga la norma IEEE 802.3 o la especificación primitiva de Ethernet.
- Datos LLC.- Unidad de datos proporcionada por el LLC.
- Relleno.- Octetos añadidos para asegurar que la trama sea lo suficientemente larga tal que la técnica de detección de colisiones (CD) funcione correctamente.
- Secuencia de comprobación de trama (FCS, Frame Check Sequence).- comprobación redundante cíclica de 32 bits (CRC), calculada teniendo en cuenta todos los campos excepto el de preámbulo, el SFD y a sí mismo (Stallings, 2000).

2.6.2 CONTROLADOR DE RED ETHERNET ENC28J60

El ENC28J60 es un controlador Ethernet autónomo diseñado para servir como una interfaz de red Ethernet para cualquier controlador equipado con

interfaz SPI. Reúne todas las especificaciones del estándar IEEE 802.3, está integrado por una MAC y un módulo PHY 10 base-T, soporta modo de transmisión full/half duplex y además posee un módulo de interfaz SPI capaz de funcionar con señales de clock arriba de los 20 MHz.

El controlador se constituye principalmente de siete bloques funcionales y la función de cada uno de ellos es descrita a continuación:

1. Una interfaz SPI que sirve como canal de comunicación entre el controlador maestro y el ENC28J60.
2. Registros de control los cuales son usados para controlar y monitorear al ENC28J60.
3. Un buffer de memoria RAM de puerto doble para los paquetes de datos recibidos y transmitidos.
4. Un árbitro para controlar el acceso al buffer RAM cuando peticiones son hechas desde el controlador DMA (Direct Memory Access Controller, Controlador de Acceso Directo a Memoria) o desde los bloques de transmisión y recepción.
5. La interfaz de bus que interpreta datos y comandos recibidos vía interfaz SPI.
6. El modulo MAC que implementa el estándar IEEE 802.3.
7. El módulo PHY (Physical Layer, Capa Física) que codifica y decodifica los datos análogos que se encuentran presentes en la interfaz de par trenzado.

En la Figura 2.11 se muestra como puede ser integrado el chip ENC28J60 a cualquier proyecto, donde tal chip solo proporciona el control sobre las dos capas inferiores del modelo de comunicación que se haga uso, es decir la capa física y la capa de acceso al medio, módulo PHY y módulo MAC respectivamente (Microchip, 2006-2012, 2010).

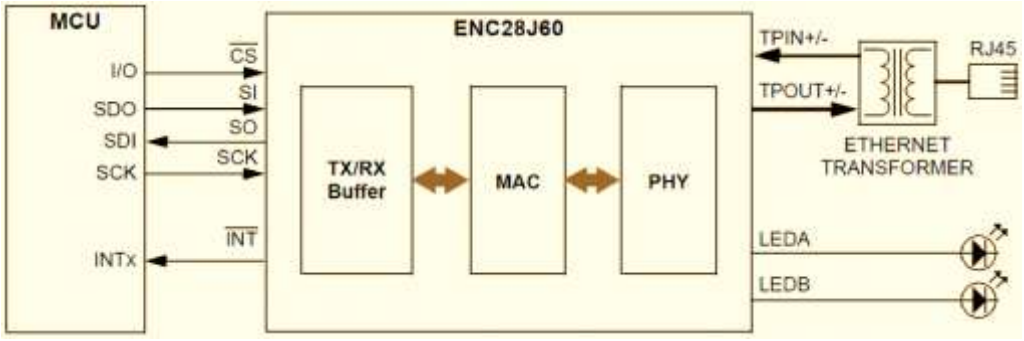


Figura 2.11.- Interfaz típica basada en el ENC28J60.

2.7 COMUNICACIÓN SERIAL RS-485

En 1983, la EIA (Electronics Industries Association, Asociación de Industrias de la Electrónica) aprobó un nuevo estándar de transmisión equilibrada llamado RS-485, el cual encontró una amplia aceptación y uso en aplicaciones industriales, médicas, así como en aplicaciones de consumo, desde entonces ha venido ganando preferencia como interfaz de uso en la industria.

Cuando se necesita transmitir a largas distancias o a velocidades más altas que las que maneja RS-232, la interfaz RS-485 es la solución, al utilizar enlaces de RS-485 no hay limitación a conectar tan solo dos dispositivos ya que dependiendo de la distancia, velocidad de transferencia y los circuitos integrados que se utilizan, se pueden conectar hasta 256 nodos con un simple par de cables.

La interfaz RS-485 posee varias ventajas en comparación con RS-232, entre las cuales se mencionan:

a) Bajo costo: Los ICs (Integrated Circuit, Circuito Integrado) para trasmitir y recibir son baratos y requieren solo de una fuente de +5V de alimentación para poder generar una diferencia mínima de 1,5V entre las salidas diferenciales. En contraste las salidas mínimas de RS-232 que son de ± 5V por lo cual se requiere de fuentes duales o IC caros que puedan generar dicha alimentación.

b) Capacidad de red. RS-485 es una interfaz que no se limita a la conexión de 2 dispositivos ya que es una interfaz multipunto con la capacidad de poder tener múltiples transmisores y receptores. Con receptores de alta impedancia, los enlaces con RS-485 pueden llegar a tener hasta 256 nodos.

c) Longitud de enlace. En un enlace RS-485 puede tener hasta aproximadamente 1220 metros de longitud, comparado con RS-232 que tiene unos límites típicos de 15 a 30 metros.

d) Rapidez. La razón de transferencia de bits puede ser tan alta como de 10 Megabits/segundo.

2.7.1 BALANCEO Y DESBALANCEO DE LINEAS

La razón principal por la cual RS-485 puede transmitir a largas distancias es debido a que utiliza el balanceo de líneas. Cada señal tiene dedicado un par de cables, sobre uno de ellos se encontrará un voltaje y en el otro estará su complemento (véase Figura 2.12), así de esta manera el receptor responde a la diferencia entre estos voltajes.

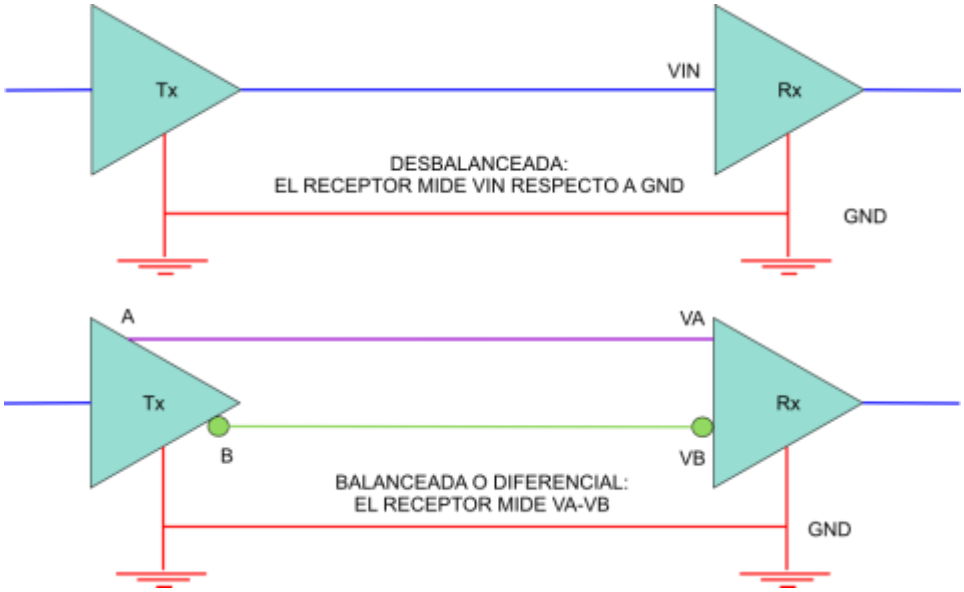


Figura 2.12.- Líneas balanceadas y desbalanceadas.

La gran ventaja de las líneas balanceadas en comparación con las no balanceadas es su inmunidad al ruido.

En cuanto a las líneas balanceadas presentadas en la Figura 2.12, la TIA/EIA-485 designa a estas dos líneas como A y B. En el controlador que transmite, una entrada alta TTL causa que la línea A sea más positiva que la línea B, mientras que un bajo en lógica TTL causa que la línea B sea más positiva que la línea A. Por otra parte en el receptor, si la entrada A es más positiva que la entrada B, la salida lógica TTL será en alto y si la entrada B es más positiva que la entrada A, la salida lógica TTL será en bajo, donde la máxima entrada diferencial (V_A - V_B) no debe ser más grande que ±6V.

2.7.2 REQUERIMIENTOS DE VOLTAJE Y DE CORRIENTE

Las interfaces RS-485 típicamente utilizan una simple fuente de +5V de alimentación, pero los niveles lógicos de los transmisores y receptores no operan a niveles estándares TTL de +5V o voltajes lógicos CMOS. Para una salida válida, la diferencia entre las salidas A y B debe ser de al menos 1,5V. Si la interfaz está perfectamente balanceada las salidas tienen un offset igual a un medio del voltaje de la fuente de alimentación, cualquier desbalance sube o baja el valor.

En el receptor RS-485 la diferencia de voltaje entre las entradas A y B necesita ser solo de 0,2V, es decir que si A es al menos 0,2V más positiva que B, el receptor ve un volar lógico de 1 y si B es al menos 0,2V más positivo que A, el receptor ve un valor lógico de 0. Si la diferencia entre A y B es menor a 0,2V el nivel lógico es indefinido.

La diferencia entre los requerimientos del transmisor y el receptor pueden tener como mínimo un margen de ruido de 1,3V. La señal diferencial puede ser atenuada o tener picos hasta de 1,3V, y aun así el receptor vera el nivel lógico correcto. El margen de ruido es menor que el de un enlace RS-232, no se debe

olvidar que RS-485 maneja señales diferenciales y que cancela la mayoría del ruido a través de su enlace.

El total de corriente utilizada por un enlace RS-485 varía con las impedancias de los componentes en el enlace, incluyendo los transmisores, receptores, cables y los componentes de terminación de enlace. Una baja impedancia a la salida del transmisor en conjunto con una baja impedancia en los cables habilita los cambios rápidos de nivel y asegura que el receptor vea la señal lo más larga posible.

Los componentes de terminación, cuando son usados, tiene un gran efecto sobre la corriente en el enlace muchos enlaces con RS-485 tienen una resistencia de 120 Ω a través de las líneas diferenciales en cada uno de los dos finales del enlace.

2.7.3 LA COMUNICACIÓN SERIAL RS-485 EN MODO FULL DUPLEX

La interfase RS-485 está diseñada para uso en sistemas multipunto, esto significa que los enlaces pueden llegar a tener más de un transmisor y receptor, ya que cada señal en el enlace (transmisión y recepción) tienen su propia ruta (véase Figura 2.13).

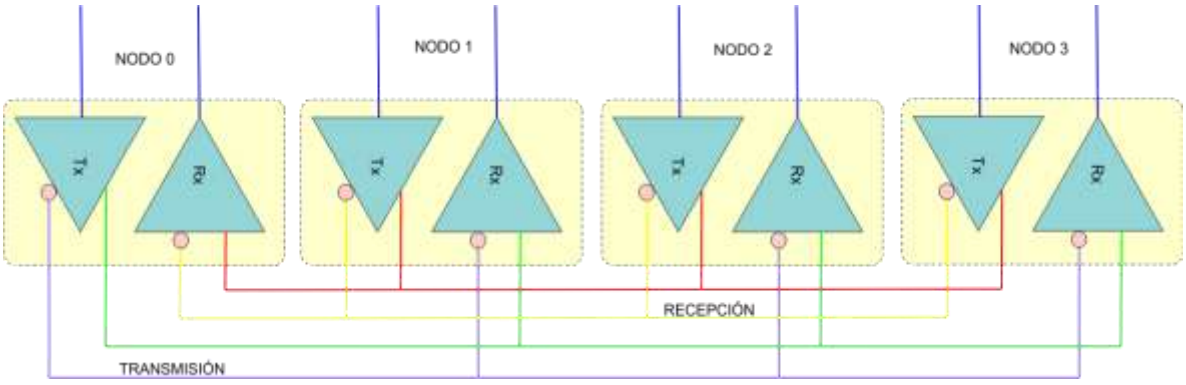


Figura 2.13.- Modo de transmisión full-duplex. Fuente: Serial Port Complete Programming and Circuits for RS-232 and RS-485 Links and Networks, Axelson (1998).

En la comunicación Full duplex todos los esclavos deben leer lo que el maestro envía, pero solo el esclavo en cuestión responde y lo hace a través de los cables opuestos.

2.7.4 LA COMUNICACIÓN SERIAL RS-485 EN MODO HALF DUPLEX

En este tipo de enlace se tienen múltiples transmisores y receptores compartiendo una misma ruta para ambos, cuando el enlace tiene 3 o más nodos, usualmente el sistema solo permite transmitir o recibir información en determinado momento, sin embargo no puede ejecutar las dos acciones al mismo tiempo. El permitir transmitir o recibir información en este tipo de enlace se puede lograr con la adicción de una señal de control, tal como se muestra en la Figura 2.14.

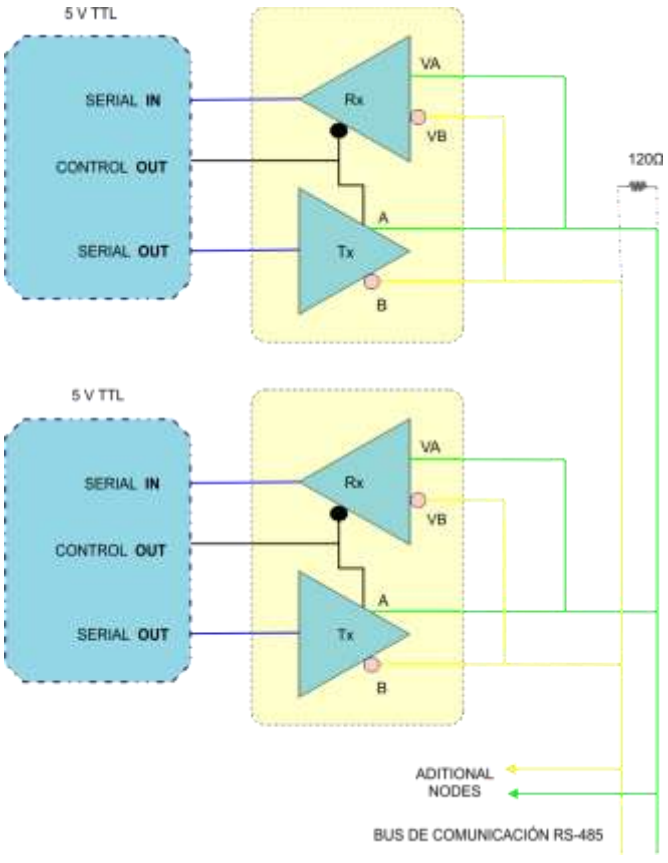


Figura 2.14.- Modo de transmisión half-duplex.

2.8 PROTOCOLO MODBUS

Modbus es un protocolo estándar que puede gestionar una comunicación tipo maestro-esclavo entre distintos equipos conectados físicamente en un bus serie. Este protocolo fue ideado para los PLCs Modicon (marca que ahora pertenece a Schneider Electric) en 1979 y con el tiempo se ha convertido en un protocolo muy empleado en las comunicaciones industriales. Las principales razones de ello son la sencillez del protocolo, versatilidad, y que sus especificaciones gestionadas por la *Modbus Organization*, son de acceso libre y gratuito.

En una red Modbus se dispone de un equipo maestro que puede acceder a varios equipos esclavos, para ello cada esclavo de la red se identifica con una dirección única de dispositivo. Para intercambiar las peticiones y respuestas, los dispositivos de una red organizan los datos en tramas. Dado que Modbus es un protocolo de nivel de aplicación se requiere utilizarlo sobre una pila de protocolos que resuelva los temas específicos del tipo de red empleada. En función de la arquitectura de protocolos usada se distinguen tres tipos de Modbus: *RTU, ASCII y MODBUSTCP*.

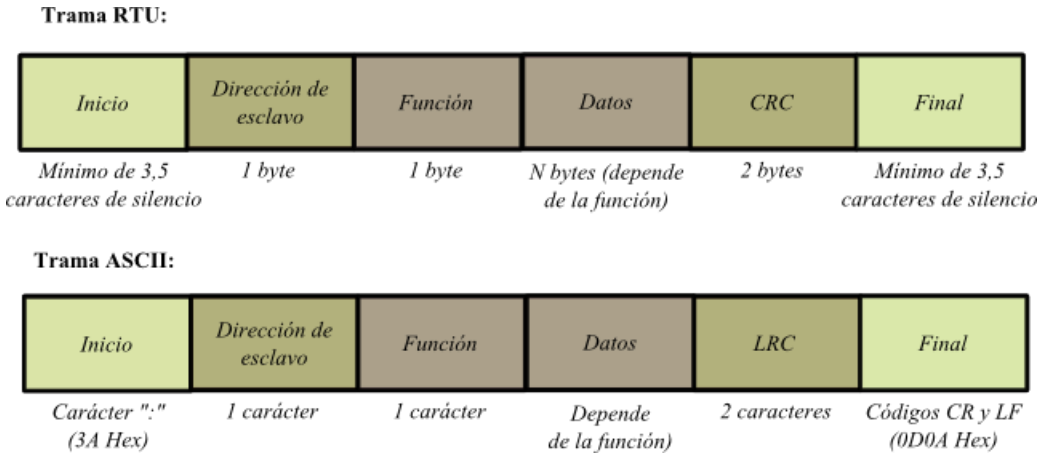


Figura 2.15.- Formato de tramas de Modbus serie.

Modbus RTU y ASCII están pensados para ser utilizadas directamente sobre un medio físico serie asíncrono, como por ejemplo EIA/TIA RS-232, EIA/TIA

RS-485 o EIA RS-422. En contraste MODBUSTCP está desarrollado para funcionar sobre redes que utilizan la arquitectura TCP/IP por lo que permite usar Modbus sobre redes como Ethernet o Wi-Fi. La trama de datos del protocolo Modbus RTU y ASCII, se muestra en la Figura 2.15.

Los campos *función* y *datos* representan la trama de nivel de aplicación de Modbus y dependen de las distintas opciones de peticiones y respuestas. El tamaño del campo de datos siempre depende de la función utilizada. La *dirección* es un valor que debe identificar unívocamente a un dispositivo esclavo de la red (Candelas, 2011).

2.9 INTERFAZ PERIFÉRICA SERIE SPI

Es una interfaz síncrona maestro-esclavo que se basa en un registro de desplazamiento de 8 bits, donde el maestro SPI genera una señal de reloj usada por todos los dispositivos SPI esclavos para coordinar la transferencia de datos (Véase Figura 2.16), el dispositivo periférico conectado al bus SPI también incluye un registro de desplazamiento, así juntos los 2 registros de desplazamiento de 8 bits son conectados de modo que generan un registro de rotación a la izquierda de 16 bits.

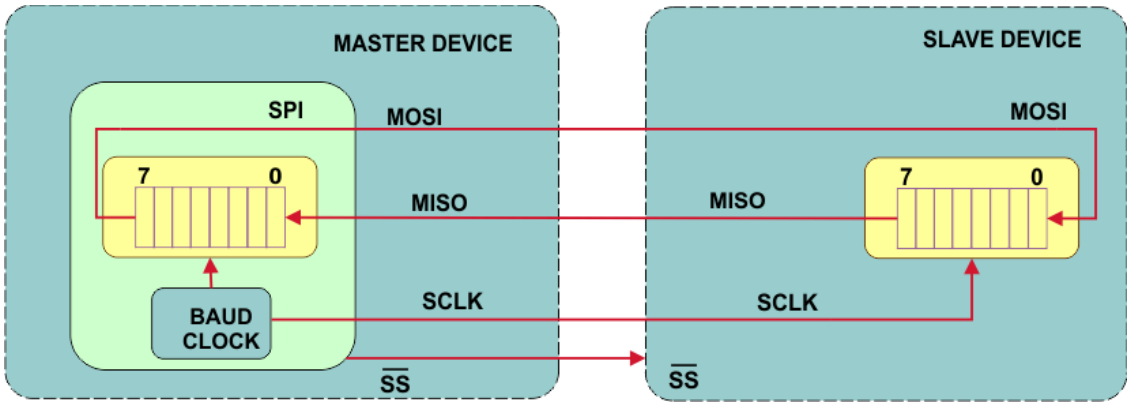


Figura 2.16.- Diagrama de conexión Maestro-Esclavo. Fuente: Descripción General de un Microcontrolador (Módulos de Comunicación), Osio *et al.* (2011).

Para el protocolo una transferencia de datos consiste de un desplazamiento de 8 bits, el cual resulta en una transferencia de datos entre el dispositivo maestro y el esclavo. Muchos dispositivos, tales como los conversores A/D, conversores D/A y Chips de memoria Flash, SRAM, SD, así como microcontroladores comerciales, etc., poseen interfaz SPI (Osio *et al.*, 2011).

2.10 COMUNICACIÓN INALÁMBRICA

La comunicación inalámbrica se da cuando los medios de unión entre sistemas no son llevados a cabo mediante cables, para lo cual sus principales ventajas son que permiten una facilidad de emplazamiento y reubicación, evitando así la necesidad de establecer un cableado y adquiriendo con ello una mayor rapidez en su instalación. Las técnicas utilizadas son: por Infrarrojos (IR) y por radio frecuencia (RF).

En relación a las comunicaciones por radio frecuencia existen factores que influyen de manera directa en su rango de trabajo, es decir que si este aumenta o disminuye depende de:

- Frecuencia.
- Potencia de salida.
- Sensibilidad de recepción.
- Características de la antena.
- Entorno de trabajo (Mayné, 2009).

2.10.1 CHIP TRANSDUCTOR RF: nRF24L01+

El nRF24L01+ es un chip transceptor simple de 2,4 GHz con un motor de protocolo en banda base embebido (Enhanced ShockBurst™), apropiado para aplicaciones inalámbricas de muy baja potencia y además también diseñado para

operar en la banda de frecuencia mundial ISM (Industrial Scientific and Medical, Industrial Científica y Medica) que cubre desde los 2,400 a los 2,4835 GHz.

Se puede operar y configurar el nRF24L01+ a través de la interfaz periférica serie (SPI), la cual permite acceder de manera directa al mapa de registros donde se encuentran los registros de configuración del chip, entre otros.

El motor de protocolo en banda base embebido es basado en paquetes de comunicación y soporta varios modos de operación, desde operación manual hasta operación como protocolo autónomo avanzado. Los buses de transmisión internos (FIFOs) aseguran el buen flujo de datos entre la parte de la comunicación por radio y el controlador maestro del sistema. Enhanced ShockBurst™ reduce los costos del sistema al manejar a altas velocidades todas las operaciones de su capa de enlace al medio (Figura 2.17).

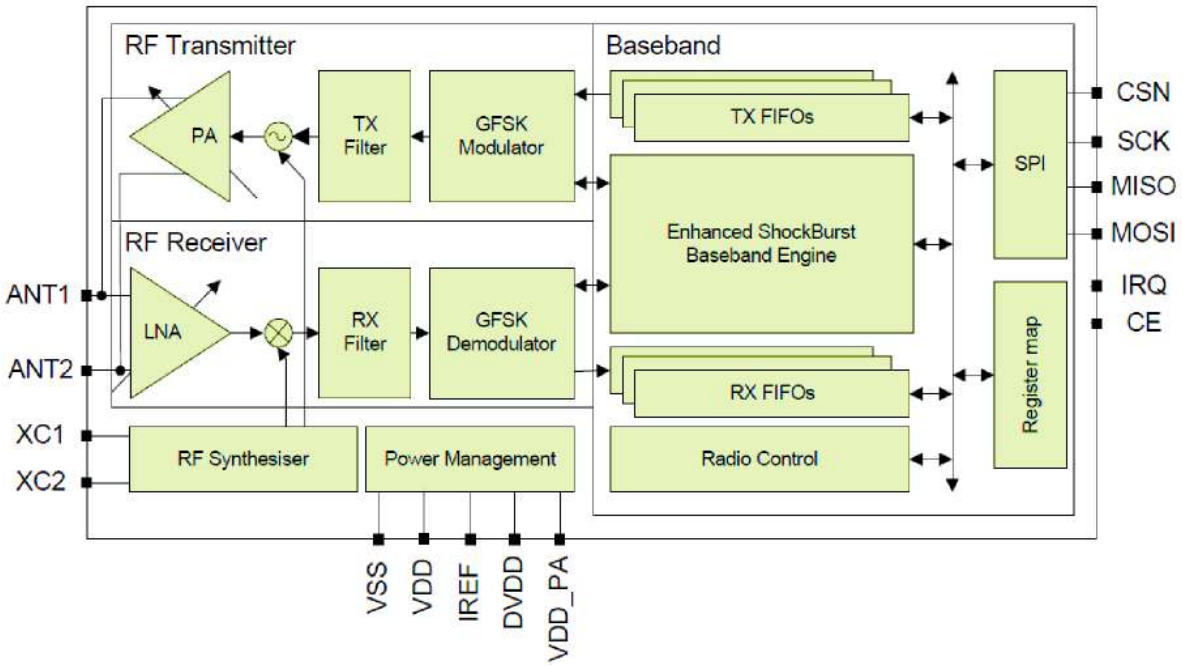


Figura 2.17.- Diagrama a bloques del chip nRF24L01+. Fuente: nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification V1.0. (Nordic Semiconductor, 2008).

La parte de la comunicación por radio utiliza modulación GSFK y algunos parámetros de ésta pueden ser configurados por el usuario, como lo son: canal de

frecuencia, potencia de salida y tasa de transferencia en el aire, donde el chip nRF24L01+ soporta tasas de transferencia de 250 Kbps, 1 Mbps y 2 Mbps, además de poseer modo de operación de ahorro de energía.

Algunas de sus características más importantes que posee el chip aparte de las ya mencionadas, son: 126 canales RF, filtros de canal integrados, sintetizadores completos integrados, el sintetizador RF acepta cristal de bajo costo de 16 MHz, soporta la transmisión de datos de entre 1 y 32 bytes, posee coordinación de paquetes automática, 6 terminales de datos para redes en estrella de 1:6 haciendo uso de MultiCeiver^{TM}, contiene integrados reguladores de voltaje y el rango de voltaje de alimentación oscila entre 1,9 a 3,6 V.

2.10.2 CONTROL DE LA RADIO

El chip nRF24L01+ se encuentra estructurado por una máquina de estados que controla las transiciones entre sus modos de operación, para ello toma como entradas los valores de los registros que el usuario define, así como también algunas señales internas. Se puede configurar el chip en 4 diferentes modos:

- Power Down Mode (modo ahorro de energía).- En este modo el chip es deshabilitado haciendo uso de un consumo mínimo de corriente del sistema.
- Standby Modes (modos de espera).- Posee dos modos de espera donde:
  a) Standby-I mode.- El chip entra a este modo para minimizar el consumo de corriente promedio mientras se mantienen tiempos cortos de puesta en marcha.
  b) Standby-II mode.- En este modo se consume más corriente en comparación con el otro modo de espera, el chip entra a este

modo si el pin *CE* es mantenido en alto en un dispositivo transmisor primario (PTX) que tiene un buffer de transmisión vacía (TX FIFO).

- RX Mode (modo de recepción).- Este es un modo activo donde la radio nRF24L01+ es usada como receptor. En este modo el receptor demodula las señales del canal RF y las presenta constantemente como datos demodulados al motor de protocolo en banda base, este a su vez busca un paquete valido y si lo encuentra (al relacionar las direcciones y CRC valido) la carga útil recibida es presentada en espacio vacante del buffer de recepción, pero si este se encuentra lleno entonces el paquete recibido es descartado.
- TX Mode (modo de transmisión).- Es un modo activo para la transmisión de paquetes. El chip permanece en este estado hasta que finalice de transmitir un paquete, si el buffer de transmisión no está vacío el dispositivo permanece en el modo de transmisión y transmite el siguiente paquete.

2.10.3 CANAL DE FRECUENCIA RF

La frecuencia del canal RF determina el centro del canal utilizado por el nRF24L01+, donde el canal ocupa un ancho de banda de menos de 1 MHz a la velocidad de 250 Kbps y 1 Mbps, y un ancho de banda menor de 2 MHz a 2 Mbps. El chip puede operar en frecuencias desde 2,400 GHz hasta 2,525 GHz.

La frecuencia del canal es asignada de acuerdo a la fórmula:

$$F_0 = 2400 + RF\_CH\ [MHz]$$

Para lo cual se debe de programar al transmisor y al receptor con la misma frecuencia de canal RF.

2.10.4 ENHANCED SHOCKBURST™

El protocolo en banda base es un tipo de paquete basado en la capa de enlace de datos del dispositivo nRF24L01+ que reúne las características de ensamblado y temporización de paquete, así como el reconocimiento automático y retransmisión de paquetes. Enhanced ShockBurst™ habilita la implementación de comunicación de muy baja potencia con un alto desempeño en sistemas de comunicación ya sean unidireccionales o bidireccionales.

El formato del paquete contiene un preámbulo, dirección, campo de control de paquete, carga útil y un campo CRC, tal paquete es descrito de mejor manera mediante la Figura 2.18.

Figura 2.18.- Paquete Enhanced ShockBurst™ con carga útil (0-32 bytes).

- Preámbulo.- Es una secuencia de bits utilizado para sincronizar al demodulador con la cadena de bits que va a recibir.
- Dirección.- Contiene la dirección del receptor del paquete, para así asegurar que el paquete sea recibido por el receptor correcto y puede tener un tamaño de 3 a 5 bytes.
- Campo de control de paquete.- Este campo a su vez se subdivide en tres secciones:
  a) Longitud de carga útil.- Es un campo de 6 bits que especifica la longitud de la carga útil en bytes.
  b) Identificación de paquete (PID).- Los 2 bits de este campo en conjunto con el campo CRC son usados para detectar si el paquete recibido es nuevo o retransmitido.
  c) Bandera de sin reconocimiento.- Esta bandera es usada solamente cuando la característica de auto reconocimiento es

usada. El poner esta bandera en alto le indica al receptor que el paquete no se auto reconoce.

- Carga útil.- Es el contenido del paquete definido por el usuario y puede poseer un tamaño de 0 a 32 bytes, es transmitido a través del aire cuando es cargado al nRF24L01+.
- Campo CRC.- Es el manejador del mecanismo de detección de errores en el paquete y puede ser de 1 o 2 bytes.

2.10.5 MULTICEIVER™

Es una característica usada al configurar al dispositivo en modo de recepción (RX), la cual contiene un conjunto de seis canales de datos paralelos con direcciones únicas. Un canal de datos es un canal lógico en el canal físico RF, así cada canal tiene su propia dirección física decodificada en el chip nRF24L01+.

El dispositivo configurado como receptor primario (PRX) puede recibir datos direccionados de 6 diferentes canales de datos, haciendo uso de un solo canal de frecuencia como se muestra en la Figura 2.19.

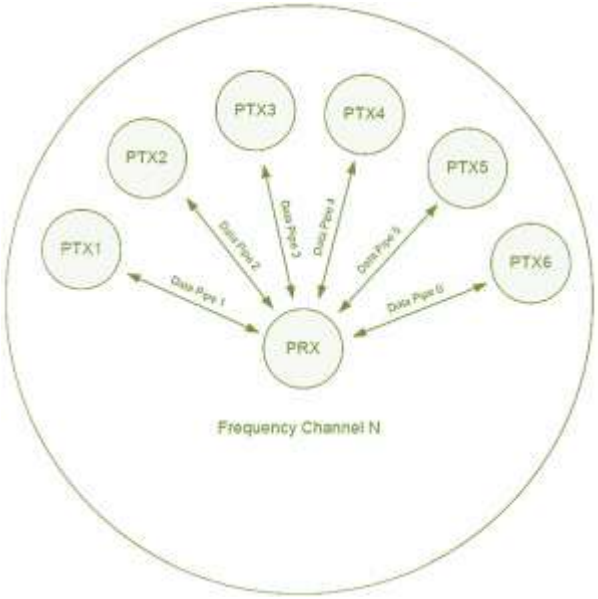


Figura 2.19.- Modo receptor de paquetes utilizando MultiCeiver™

2.11 SMART SENSORS

El termino *smart sensor (*sensor inteligente*)* fue acuñado a mediados de la década de 1980 y desde entonces a varios dispositivos se les ha llegado a nombrar smart sensors. La inteligencia requerida por tales dispositivos puede ser obtenida a partir de unidades de microcontrolador (MCU), procesadores de señales digitales (DSP), y circuitos integrados de aplicación específica (ASIC), las cuales son tecnologías desarrolladas por varios fabricantes de semiconductores.

La definición de smart sensor no ha sido tan ampliamente aceptada, por lo cual es objeto de discusión. Sin embargo, la especificación IEEE 1451.2 define a un smart sensor como “un sensor que provee funciones más allá de las necesarias para generar una correcta representación de una cantidad sensada o controlada” (Frank, 2000).

De igual forma Rivera *et al.* (2008), mencionan que usando las referencias con respecto a intelligent sensors y la definición de smart sensor del IEEE, la clasificación de inteligencia en los sensores se puede llevar a cabo de acuerdo a sus funcionalidades (véase Tabla 2-1). Donde puede observarse que para que un sensor pueda ser considerado smart sensor debe conjuntar como mínimo las etapas de procesamiento, comunicación e integración.

Tabla 2-1.- Clasificación de inteligencia en los sensores. Fuente: (Rivera *et al.,* 2008*).*

SENSOR FUNCTIONALITIES		
Processing Communication Integration	Compensation Processing Communication Integration	Compensation Processing Communication Integration Validation Data Fusion
SMART SENSORS	**INTELLIGENT SENSOR**	**+INTELLIGENT SENSOR**

3.1 PLANTEAMIENTO METODOLÓGICO

En complemento con la información que se proporcionó en la sección de planteamiento general del capítulo 1 (sección 1.5), las 5 partes principales del desarrollo del proyecto se pueden visualizar a través de la Figura 3.1.

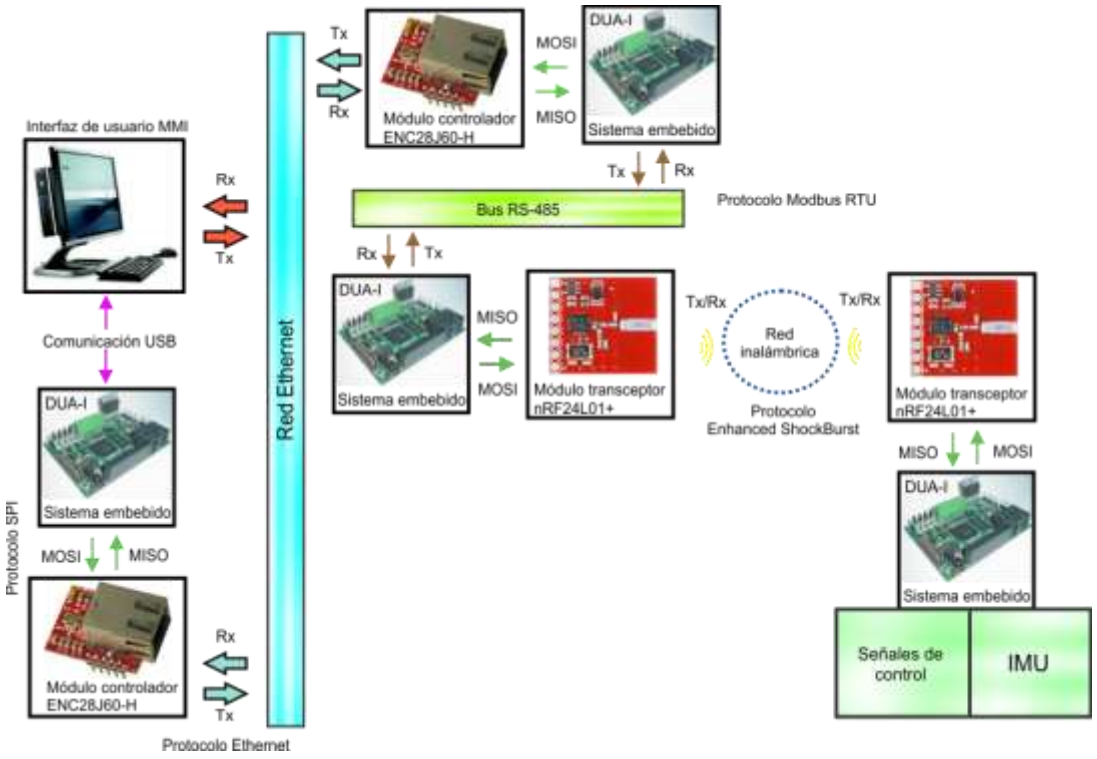


Figura 3.1.- Planteamiento metodológico general.

Debido a que el proyecto principalmente se centra en cubrir los aspectos de comunicación de todo un sistema y que además el sistema embebido, encargado mediante protocolo SPI de controlar la tarjeta ENC28J60-H, no contaba con el software para lograr la conexión a la red Ethernet, como primer punto se debían ***desarrollar librerías de software*** que permitieran la conexión de tal manera que no solo se pudieran transferir datos dentro de la red a nivel de *capa de internet* sino que además se permitiera el gestionamiento de conectividad por parte de un ordenador (flechas rojas en Figura 3.1).

Para complementar el sistema de integración y así lograr los enlaces de comunicación a través de Modbus RTU y Enhanced ShockBurst™ que se muestran a la derecha de la red Ethernet en la Figura 3.1, se debía continuar con el estudio del ***manejo de software para estos protocolos.***

Otra de las características aún no mencionadas sobre la tarjeta DUA-I es la de tener integrada una Unidad de Medición Inercial (IMU, Inertial Measuring Unit) basada en MEMS que permite una amplia variedad de aplicaciones. La unidad integra un giroscopio, un acelerómetro, un magnetómetro y un sensor de temperatura. Al hacer usó de la IMU para las cuestiones de monitoreo, se debía llevar a cabo la ***revisión y selección de sensores*** necesarios para mostrar y analizar parámetros de rendimiento en torno a los protocolos de comunicación utilizados.

También se desarrollaría una ***interfaz hombre-máquina (MMI)*** o sistema *SCADA* con el fin de poder visualizar datos y enviar señales de control a través de la red hacia el proceso sujeto a control y monitoreo, donde dicha interfaz se comunicaría mediante el bus serie universal con el sistema embebido.

Por ultimo de igual manera a través de la Figura 3.1 es posible observar de forma clara cómo interactúan los diversos dispositivos entre sí, desde el punto de inicio hasta el punto final, se presentó de esta forma con el fin de que la estructura de red mostrada pudiera ser implementada sobre un sistema físico (proceso), lo cual culminaría con la ***implementación de la red multiprotocolo***.

3.2 DESARROLLO DE LIBRERÍAS

El desarrollo de librerías de software para su integración al entorno xQIDE consiste en la generación de código fuente que permita acceder al módulo hardware desarrollado y conectado al sistema embebido, y son implementadas principalmente mediante un archivo de cabecera **.h** y un archivo de

implementación de funciones **.c**. Estos archivos como menciona Morales (2013), “requieren cierto nivel de estructura para facilitar su uso” y para ello proporciona un conjunto de recomendaciones para su implementación en ocho principales secciones de código.

Este proyecto al desarrollarse en base a un módulo hardware ya integrado, requería la creación de un archivo de cabecera donde se debían realizar nuevas definiciones del mapeo de puertos por defecto, de la declaración de los puertos requeridos por el módulo, de banderas de control, de declaraciones de macros, de identificadores para el manejador de interrupciones, de la declaración de los prototipos de función y de inspectores de depuración. En este caso se partió de un archivo de cabecera pre-estructurado y agregado al entorno xQIDE llamado *ethernet.h* que contenía la mayor parte de lo mencionado anteriormente y con lo cual se daba pleno manejo de la comunicación SPI a través de las siguientes funciones:

eth_reset_low().- Encargada de mantener en un estado lógico bajo la señal de reset que envía el módulo.

eth_reset_high().- Desempeña la función contraria a la función *eth_reset_low()*.

eth_cs_low().- Esta función mantiene en un estado bajo la señal *chip select*, habilitado así la transferencia de datos hacia el dispositivo esclavo.

eth_cs_high().- Mantiene la señal *chip select* en un estado alto, con lo cual la comunicación mediante interfaz SPI es finalizada.

eth_send_byte().- El módulo SPI con el que cuenta el sistema embebido es realizado principalmente para trabajar en la comunicación como maestro, es decir que se comporta como unidad de control. Al ser el dispositivo maestro mediante esta función el hardware coloca sobre la línea MOSI (*Master Out Slave In, Salida del Maestro Entrada del Esclavo*) el dato de 8 bits alojado en su argumento.

eth_get_byte().- Esta función le indica al módulo hardware que proporcione el dato de 8 bits que fue leído sobre la línea MISO (*Master In Slave Out, Entrada del Maestro, Salida del Esclavo*) durante la transferencia de datos.

La tarjeta controladora de red Ethernet basa su funcionamiento en el dispositivo ENC28J60 de Microchip®, el cual es un controlador Ethernet autónomo compatible con el estándar IEEE 802.3. Al integrarse el dispositivo a un proyecto se le proporciona al diseñador un medio para implementar la interconectividad de sistemas a través de Ethernet pudiendo seguir la arquitectura de la familia de protocolos TCP/IP, en este proyecto se hizo uso de tal arquitectura. Al ser el ENC28J60 el componente principal para la comunicación a través de Ethernet, el desarrollo de software se centró en la información obtenida mediante la hoja de especificaciones, la hoja de errores del dispositivo (Microchip, 2006-2012, 2010) y la información presentada en el manuscrito anónimo y no publicado que lleva por nombre *Módulo Ethernet ENC28J60 para microcontroladores PIC*.

La revisión de la organización de memoria del ENC28J60 fue un punto clave para entender su funcionamiento (Figura 3.2), la cual se trata de una memoria RAM estática dividida en tres principales memorias:

- Los registros de control.- Este espacio de memoria a su vez se divide en 4 bancos constituidos por registros de 8 bits, agrupados como registros: ETH, MAC y MII, y son usados para la configuración, control y pruebas de estado del dispositivo.
- El buffer Ethernet.- Es el espacio que contiene la memoria usada por el dispositivo, tanto para el buffer de transmisión así como para el buffer de recepción y su tamaño es de 8 Kbytes.
- Los registros PHY.- Son usados para configuración, control y pruebas de estado del módulo PHY, se accede a ellos a través del manejador de interfaz media independiente (MIIM) implementado en la MAC.

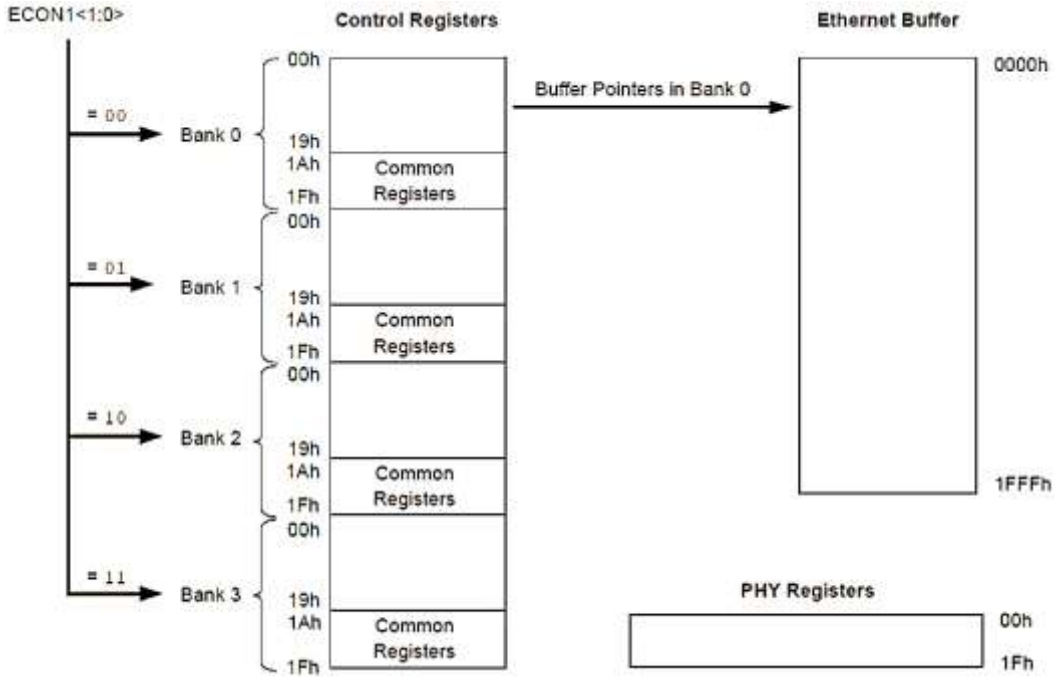


Figura 3.2.- Organización de memoria del ENC28J60.

La operación del ENC28J60 depende de una serie de comandos recibidos a través de la interfaz SPI (Tabla 3-1). Los comandos son instrucciones de uno o más bytes necesarios para acceder a la memoria de los registros de control y al espacio de memoria del buffer Ethernet, como mínimo se componen de 3-bits de cabecera seguidos por un argumento de 5-bits que especifica una dirección de registro o un dato constante. Las instrucciones de escritura y campo de bits (write & bit field) son también seguidas por uno o más bytes de datos.

Tabla 3-1.- Conjunto de instrucciones de configuración para el ENC28J60.

Instrucción Nombre y Mnemónico	**Byte 0**		**Byte 1 y siguientes**
	opcode	Argumento	Dato
Read Control Register (RCR)	0 0 0	a a a a a	N/A
Read Buffer Memory (RBM)	0 0 1	1 1 0 1 0	N/A
Write Control Register (WCR)	0 1 0	a a a a a	d d d d d d d d
Write Buffer Memory (WBM)	0 1 1	1 1 0 1 0	d d d d d d d d
Bit Field Set (BFS)	1 0 0	a a a a a	d d d d d d d d
Bit Field Clear (BFC)	1 0 1	a a a a a	d d d d d d d d
System Reset Command (SRC)	1 1 1	1 1 1 1 1	N/A
Leyenda: a=Dirección de Registro de Control, d= Carga útil			

Conociendo esta información se trabajó solo en la sección de *prototipos de función* del archivo *ethernet.h* debido a que los desarrollos mediante el presente no formaron como tal, parte principal del manejo del módulo SPI a nivel hardware sino que solo se tuvo acceso a la comunicación con el fin de enviar comandos al ENC28J60. El archivo de cabecera se puede apreciar de manera completa en la sección 6.1.1 del apéndice, más adelante a lo largo de su descripción solo se presentan fragmentos del mismo.

Para una manipulación flexible y rápida de los comandos, de cantidades constantes, de las direcciones de los distintos registros de control utilizados (ETH, MAC, MII y PHY), entre otros, se declararon todos ellos como *macros*.

```
/* Function prototypes */
// ENC28J60 macro definitions
// command codes
#define WCR (0x40)
#define BFS (0x80)
#define BFC (0xA0)
#define RCR (0x00)
#define RBM (0x3A)
#define WBM (0x7A)
#define SRC (0xFF)
// communication modes
#define FULL_DUPLEX_MODE 1
#define HALF_DUPLEX_MODE 0
// used ENC28J60 control register map
#define ECON1     (0x1F)
#define ECON2     (0x1E)
#define EIR       (0x1C)
#define EREVID    (0x12)
     ....
```

Los códigos de comando se definieron de manera que se contará con el *opcode* del comando en cuestión y el argumento en ceros o con el dato fijo respectivo (en ceros para poder aplicar operación lógica OR). Para las direcciones de los registros solo se relacionó el nombre del registro con su respectiva dirección en formato hexadecimal. En el caso de las banderas *FULL_DUPLEX_MODE* y *HALF_DUPLEX_MODE* se crearon para poder elegir el modo de configuración de comunicación full o half dúplex del dispositivo.

```
    ....
// sizes of the memory areas
#define RX_BUFF_START   (0x00)
#define RX_BUFF_END     (0x19F3)
#define TX_BUFF_START   (0x19F4)
// protocol flags
#define PROTOCOLO_ARP   (0x806)
#define PROTOCOLO_IP    (0x800)
#define PROTOCOLO_ICMP  (0x01)
#define PROTOCOLO_TCP   (0x06)
#define PROTOCOLO_UDP   (0x11)
#define PROTOCOLO_TMP   (0x04)
// implementation to select bank of memory
#define s_bank(x) c_reg(ECON1,3); \
                  s_reg(ECON1,x)
// implementation to reset the ENC28J60
#define reset()   eth_cs_low(); \
                  eth_send_byte(SRC); \
                  eth_cs_high()
    ....
```

Para configurar el buffer Ethernet es necesario delimitar la parte que se ocupa de la memoria para el buffer de recepción y para el buffer de transmisión, la distribución puede darse por parte del diseñador de manera aleatoria pero las longitudes deben tenerse en cuenta dependiendo los posibles casos de uso, es decir que si se espera la llegada de una gran cantidad de paquetes se puede destinar gran parte para la recepción y una menor parte para la transmisión.

Es importante conocer que para que el buffer de recepción, el cual se comporta como una memoria circular FIFO, no sea corrompido, debe alojarse en el segmento más bajo de memoria y que además se debe asegurar que termine en una cantidad impar. Solo se configura el inicio y fin del buffer de recepción, ya que el resto de memoria para complementar los 8 Kbytes el hardware del ENC28J60 lo toma automáticamente como espacio designado para el buffer de transmisión.

De igual manera se hicieron las definiciones de los valores estandarizados de los protocolos que se utilizan a lo largo del proyecto, estos códigos son usados en diferentes campos de mensaje, por ejemplo en la trama MAC de Ethernet, así como en algunos campos de las cabeceras de protocolo como ARP e IP. El valor

del macro *PROTOCOLO_TMP* fue creado de manera aleatoria con el fin de poseer un identificador para el mensaje particular a transmitir por el sistema embebido. En el archivo de cabecera también fue posible la declaración de algunas funciones haciendo uso de macros, tales como:

s_bank().- Es la encargada de posicionar el apuntador de banco de memoria a partir de la reposición y escritura de bits del registro ETH llamado *ECON1*, al hacer su llamado es posible leer o escribir los registros del banco seleccionado. Con el fin de evitar la lectura y la escritura de registros de manera errónea, se debe asegurar que se encuentre posicionado sobre el banco de registros correcto.

reset().- Su función es la de resetear el hardware del dispositivo mediante él envió del comando SRC.

La implementación del archivo culminó con la declaración de variables globales, la declaración de prototipos de funciones que no pudieron ser declaradas como macros y la declaración de enlace con la implementación de funciones.

En el archivo *ethernet.c* se desarrollaron funciones para la implementación del conjunto de instrucciones que acepta el dispositivo, entre otras, y fue necesario revisar las distintas gráficas de secuencia de comando que se proporcionan en la hoja de especificaciones (Microchip, 2006-2012, 2010), ya que en base a ellas se logró observar de qué manera la interfaz SPI debía ser manipulada para lograr la acción en cuestión. El archivo se muestra completo en la sección 6.1.2 y la descripción de funciones desarrolladas se presenta a continuación:

void c_reg(char reg, char data).- Reposiciona a 0 los bits del registro que en la palabra *data* son igual a 1, sin afectar el valor de los bits restantes (implementa instrucción BFC).

void s_reg(char reg, char data).- Posiciona a 1 los bits del registro que en la palabra *data* son igual a 1, sin afectar el valor de los demás bits (implementa instrucción BFS).

void w_reg(char reg, char data).- Se encarga de realizar la escritura de los diferentes registros de control ETH, MAC y MII, mediante el uso del comando WCR en conjunto con la dirección del registro y el dato a ser escrito que recibe como argumentos.

char r_reg_eth(char reg).- Realiza la lectura de los registros ETH mediante el uso de la instrucción RCR. La dirección del registro la recibe como argumento y entonces retorna un dato de tipo *char* con la lectura.

char r_reg_mac_mii(char reg).- Desempeña la misma acción que la función *r_reg_eth()* solo que en este caso se implementa para la lectura de registros MAC y MII, debido a que este tipo de registros presentan sobre la interfaz SPI un dato de salida ficticio antes del verdadero.

void w_buff_memory(char data).- Implementa el comando WBM y se usa para poder escribir sobre el buffer de transmisión 1 byte de datos.

char r_buff_memory(void).- Es la encargada mediante el uso del comando RBM de realizar la lectura de 1 byte de datos alojado en el buffer de recepción.

void mw_buff_memory(char *buff, int length).- Dentro del buffer de transmisión pueden ser escritos más de un solo byte, siempre y cuando después de ejecutar la escritura del primer dato se le siga proporcionando al ENC28J60 señal de reloj, se presente dato sobre la línea MOSI y que además continúe habilitado mediante la señal *chip select*. En base a esto la función recibe como argumento la dirección de memoria en la cual comienzan los datos a ser escritos así como su cantidad.

void mr_buff_memory(char *buff, int length).- Ejecuta la lectura múltiple sobre el buffer de recepción debido a las mismas razones que la función *mw_buff_memory(),* la dirección de memoria donde comenzará el almacenamiento de datos y la cantidad de datos a leer los recibe como argumentos.

int r_reg_phy(char reg).- Lleva a cabo la lectura de los registros PHY del dispositivo, los cuales son registros de 16 bits que a diferencia de los registros ETH, MAC y MII o el buffer de memoria no son accesibles directamente a través de la interfaz SPI en cambio se accede a ellos con ayuda de los registros MII. Para lograr su función primero escribe la dirección del registro a leer dentro del registro *MIREGADR,* después coloca a 1 el bit *MIIRD* del registro *MICMD*, así la lectura del registro comienza y automáticamente se coloca en 1 el bit *BUSY* del registro *MISTAT*, por ello debe mantenerse leyendo este bit hasta que se reposicione por sí mismo, la lectura del registro termina con la reposición a 0 del bit *MIIRD* y su valor es conocido mediante la lectura de los registros *MIRDL* y *MIRDH.*

void w_reg_phy(char reg,int data).- Realiza la escritura de los registros PHY de forma parecida como lo hace la función de lectura, primero escribe la dirección del registro a escribir dentro del registro *MIREGADR,* después escribe en el registro *MIWRL* el byte menos significativo de la palabra *data,* mientras que el byte más significativo lo escribe en el registro *MIWRH,* al hacer esto la transacción y la colocación en 1 del bit *BUSY* del registro *MISTAT* comienzan automáticamente, se observa este bit hasta que se reposiciona de nueva cuenta por sí mismo con el fin de asegurar que la escritura se haya completado.

int checksum(int *data, int size).- Calcula la suma de comprobación de palabras de 16 bits que es requerida por varios protocolos industriales estándar, incluyendo los protocolos TCP e IP. Recibe como argumentos la dirección de memoria donde comienza el alojamiento de las palabras ya constituidas en 16 bits y la cantidad de las mismas. El cálculo lo realiza a nivel bit, suma las palabras una a una y sujeta los resultados a la misma longitud de palabra, cualquier acarreo de bits generado es sumado sin excepción en la parte menos significativa de la suma, por ultimo retorna el complemento a 1 del cálculo realizado.

void set_enc28j60(char mode, char *MAC_ENC).- Configura al dispositivo ENC28J60 lo más apegado al estándar IEEE 802.3, ya que antes de poder ser

utilizado para recibir o transmitir ciertos parámetros deben ser configurados después del reseteo del hardware.

Es aquí donde mediante la escritura de los registros *ERXSTL* y *ERXSTH* es configurado el inicio del buffer de recepción mientras que su final es escrito en los registros *ERXNDL* y *ERXNDH*, y también en los registros *ERXRDPTL* y *ERXRDPTH,* ya que estos últimos son apuntadores de lectura del buffer de recepción que son útiles para el cálculo del espacio libre en el buffer y también usados por el hardware para evitar la sobre-escritura de paquetes Ethernet sin procesar. El inicio del buffer de transmisión se configura mediante la escritura de los registros *ETXSTL* y *ETXSTH* (se hace en la configuración para no tener que escribirlo cada que se envíe un paquete).

En cuanto a las configuraciones de los registros MAC se siguen algunas recomendaciones de los valores usuales que son usados en los modos half y full duplex por ejemplo; para la programación de los registros *MABBIPG, MAIPGL* y *MAIPGH*, encargados de las pausas entre paquetes. También se programa el registro *MACON1* de modo que permita al módulo MAC él envió de paquetes de pausas de control y que lo habilite a la recepción de paquetes. El registro *MACON3* se programa dependiendo el modo de comunicación elegido mediante la variable *mode* que recibe la función como argumento, donde sus principales acciones son las de lograr que los paquetes cortos sean rellenados a la cantidad de al menos 60 bytes y que un CRC valido aparezca.

La programación del registro *PHCON1* se da de tal modo que el módulo PHY del dispositivo funcione en operación normal y que esta corresponda al modo de comunicación utilizado, mientras que *PHCON2* es programado para deshabilitar el diagnostico loopback debido a que no es confiable. En los registros *MAMXFLL* y *MAMXFLH* se programa la máxima longitud del paquete completo de datos a ser transmitido o recibido. La dirección MAC que será asignada al

dispositivo la recibe la función por medio del puntero *MAC_ENC* y es programada dentro de los registros *MAADR1:MAADR6.*

El registro *ERXFCON,* el cual permite configurar los filtros de recepción utilizados por el hardware para aceptar y rechazar paquetes, se configura para aceptar aquellos paquetes que tengan una dirección MAC de destino que concuerde con la del dispositivo, rechazar los paquetes que no tengan CRC valido, deshabilitar algunos filtros y habilitar la aceptación de mensajes con dirección de destino *broadcast* (dirección FF-FF-FF-FF-FF-FF). Por último se coloca en alto el bit *RXEN* del registro *ECON1* para habilitar de forma definitiva la recepción de paquetes Ethernet.

void mac_header(char *MAC_ENC, char *MAC_target, int protocol).- La dirección MAC agregada al dispositivo es la dirección de hardware que identifica al ENC28J60 dentro de la red Ethernet, pero solo es utilizada por los filtros para poder aceptar o rechazar paquetes, es decir que el dispositivo no escribe esta dirección al enviar un paquete, solo se encarga de generar los campos preámbulo, delimitador de comienzo de trama, relleno si es necesario y el chequeo cíclico redundante (CRC), así que se deben generar y escribir adicionalmente todos los campos faltantes del paquete Ethernet a transmitir.

También es necesario conocer que para transmitir un paquete Ethernet se debe saber la cantidad de bytes que integran el mensaje completo, por ello la variable global *size_buff* es la encargada de llevar el conteo de bytes escritos en el buffer de transmisión.

La función actual implementa la cabecera de la trama MAC del estándar IEEE 802.3. Después de inicializar los punteros de escritura del buffer y antes de comenzar la escritura de los campos, se escribe un byte de configuración llamado "*byte de control por paquete"* con valor de 0x00h de tal modo que con ello indica que la transmisión se llevará a cabo de acuerdo a las configuraciones del registro

MACON3, seguido se escriben sobre el buffer las direcciones MAC de destino y origen, y el campo tipo (protocolo de tercer nivel transportado).

void arp_header(char *src_IP, char *trgt_IP, char *MAC_ENC, char *MAC_target, char operation).- Esta función implementa la escritura de la cabecera del protocolo ARP ya sea para mensajes de tipo petición o de respuesta. Comienza por habilitar la escritura sobre el buffer y uno a uno los campos se integran de la siguiente manera:

- Tipo de dirección de hardware.- Como es Ethernet tiene valor 0x1h.
- Tipo de dirección de protocolo de red.- Como se implementa IP de manera automática se escribe mediante el macro *PROTOCOLO _IP*.
- Longitud de dirección de hardware.- Para Ethernet son 6 bytes.
- Longitud de dirección de protocolo.- Como se hace uso del protocolo IP en su versión 4 (IPv4), son 4 bytes.
- Operación.- Se escribe mediante el argumento *operation* y tiene dos significados, si es 1 es para formular un mensaje de petición y si es 2 es para formular un mensaje de respuesta.
- Dirección hardware del emisor.- Los 6 bytes se escriben sobre el buffer con ayuda del apuntador *MAC_ENC*.
- Dirección de red del emisor.- Se escribe con el apuntador *src_IP*.
- Dirección hardware del destinatario.- Se escribe con el apuntador *MAC_target*.
- Dirección de red del destinatario.- Se escribe mediante el apuntador *trgt_IP*.

void ip_header(int total_length, int protocol_layer, char *src_IP, char *trgt_IP).- Realizada con el fin de implementar la escritura de la cabecera del protocolo IP sobre el buffer de transmisión. Antes de comenzar la escritura de la cabecera ya se conocen todos los parámetros que la conformarán y por tal razón se inicia con el calculó de la suma de comprobación, ya obtenido el cálculo y almacenado en la variable *suma,* se habilita la escritura sobre el buffer de

transmisión y los distintos campos del protocolo se configuran de la forma siguiente:

- Versión.- Se configura con un valor de 4 debido al uso de IPv4.
- Longitud de la cabecera de internet (IHL).- Como en el proyecto no se hace uso del campo de opciones que provee el protocolo IP, la cabecera se ajusta de forma permanente a un tamaño de 5 palabras.
- Tipo de servicio.- Se programa con valor de 0 para indicar prioridad normal al enrutamiento de los datos, aunque no tiene uso debido a que el proyecto solo implementa comunicación de manera local.
- Longitud total.- Se programa con el valor de *total_length* más las 20 unidades correspondientes a la cabecera IP.
- Identificador.- Se programa con el valor de la variable estática de tipo entero llamada *ident*.
- Indicadores.- Se le asigna al indicador flag un valor de 0 debido a que no se usa, después un 1 al indicador de no fragmentar para indicar que no se puede dar la fragmentación de paquetes y por tanto el indicador de más fragmentos toma un valor de 0.
- Desplazamiento del fragmento.- Como la fragmentación no se puede dar este campo tiene valor de 0 (no existe fragmento pendiente por recibir). Este campo en complemento con los valores de indicadores generan una palabra de 16 bits con valor en formato hexadecimal de 0x4000h.
- Tiempo de vida (TTL).- Se asignan 128 segundos de vida para el paquete.
- Protocolo.- Se asigna el protocolo que transporta el datagrama IP mediante la variable *protocol_layer*.
- Suma de comprobación de la cabecera.- Se programa con el contenido de la variable suma.
- Dirección origen.- Se escribe el campo con ayuda del apuntador *src_IP*.
- Dirección destino.- Se escribe con ayuda del apuntador *trgt_IP*.

Se finaliza la escritura en el buffer y por último se incrementa en 1 la variable estática *ident* con el fin de poseer un nuevo identificador de paquete IP, cada que se haga el llamado de la función.

void icmp_header(int *data, int length, int ident_ICMP, int number_sequence).- Es la encargada de implementar la cabecera del protocolo ICMP y aunque el protocolo es capaz de manejar varios tipos de mensaje, en este proyecto se implementa solo para el tipo respuesta a eco. Al ser esta su naturaleza debe recibir por medio del apuntador *data* y los argumentos *ident_ICMP* y *number_sequence*, el contenido del campo de datos, identificador y número de secuencia de manera respectiva, del mensaje de petición de eco recibido, ya que estos campos deben ser íntegramente incluidos en el mensaje de repuesta. Antes de intestar la cabecera del protocolo, con ayuda del vector *header_icmp* se lleva a cabo mediante el mismo método que el protocolo IP el cálculo de la suma de comprobación del mensaje completo.

Se continúa al habilitar la escritura en el buffer de transmisión y los distintos campos del mensaje se escriben de la forma siguiente:

- Tipo.- Debido a que es respuesta a eco tiene un valor de 0.
- Código.- Posee valor de 0 ya sea para solicitud o respuesta a eco.
- Suma de comprobación.- Se le asigna el valor de *suma.*
- Identificador.- Obtiene el valor de *ident_ICMP*.
- Número de secuencia.- Obtiene el valor de *number_sequence*.
- Datos.- Escrito mediante el apuntador *data*.

void send_message().- Desempeña la acción de enviar a través del medio el mensaje Ethernet que se encuentre confeccionado y listo para su envío. Comienza por comprobar si no existe pendiente él envió de algún mensaje mediante la lectura del bit *TXERIF* del registro *EIR* ya que es una bandera de interrupción de transmisión que al encontrarse en alto da conocer que ocurrieron colisiones u otros factores que pueden parar la lógica del hardware de transmisión, se lee el bit

y si se detectó aborto de trasmisión se resetea y se reposiciona la lógica de transmisión para permitir él envió de nuevos paquetes.

Se continúa con la lectura del bit *TXRTS* del registro *ECON1* debido a que el controlador de acceso directo a la memoria (DMA) y el motor de transmisión comparten el mismo puerto de acceso a la memoria, y por lo tanto deben esperar hasta que uno u otro terminen de usarlo.

Después son configurados los punteros que marcan el fin del mensaje, es decir los registros ETXNDL y ETXNDH. Para terminar solo hace de nueva cuenta la colocación en alto del bit *TXRTS* del registro *ECON1* (al terminar la transmisión del mensaje se reposiciona de manera automática) para ponerse a la espera de que el medio permita él envió.

void freeing_buffer().- Al encontrarse el dispositivo ENC28J60 continuamente recibiendo paquetes Ethernet, se debe ser capaz de liberar la memoria del buffer de recepción ocupada por aquellos paquetes que hayan sido ya revisados y procesados. Tal acción es desempeñada por esta función y se implementa de modo tal que libere el espacio utilizado por el último paquete procesado. Para ello solo corrobora que el valor de *free_pointer* no salga de los límites establecidos para el buffer, después coloca en 1 el bit *PKTDEC* del registro *ECON2* para así decrementar en 1 el valor del registro *EPKTCNT* (este registro manifiesta la cantidad de paquetes que han sido recibidos exitosamente).

Por último se escriben los registros *ERXRDPTL* y *ERXRDPTH* para liberar el espacio. Se debe tener en cuenta que si no se libera memoria y el buffer es llenado completamente, todo nuevo paquete entrante será descartado.

void send_arp_package(char *my_MAC, char *my_IP, char *trgt_IP).- Esta función es la encargada de enviar un mensaje ARP de tipo petición a través de Ethernet. Hace uso de un arreglo auxiliar para enviar la dirección física de destino utilizada en la cabecera MAC ya que para este tipo de mensajes es necesaria una

dirección *broadcast*, después hace el llamado de las funciones que implementan los encabezados de los protocolos involucrados y la que efectúa él envió a través de la red, tal como se muestra en la Figura 3.3.

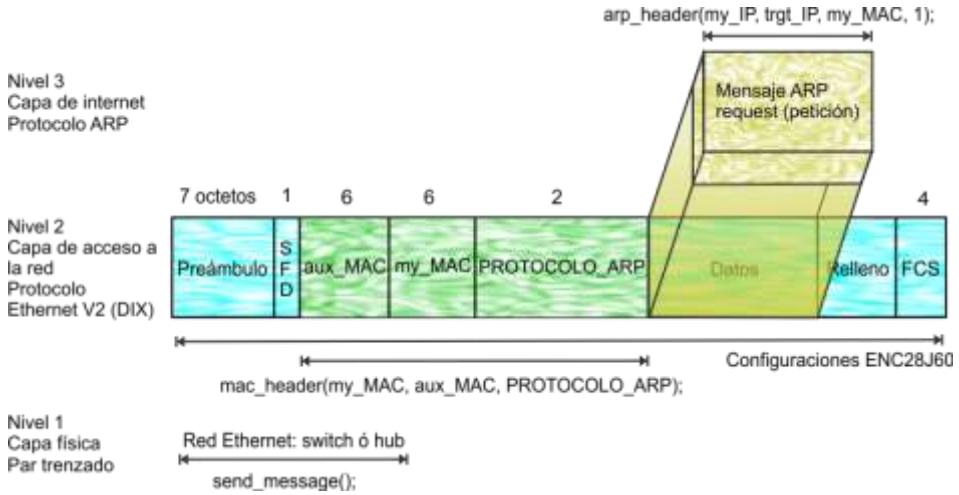


Figura 3.3.- Funcionamiento de envió de paquete ARP tipo petición.

void send_ip_package(char *data, int length, char *my_MAC, char *my_IP, char *trgt_MAC, char *trgt_IP).- Se encarga de confeccionar los paquetes IP transmitidos por el sistema embebido mediante el uso del identificador *PROTOCOLO_TMP*. Para ello después de llevar a cabo la intestación de la cabecera IP, los datos a enviar son escritos de manera subsecuente haciendo uso de la función *mw_buff_memory(),* tal como se muestra en la Figura 3.4.

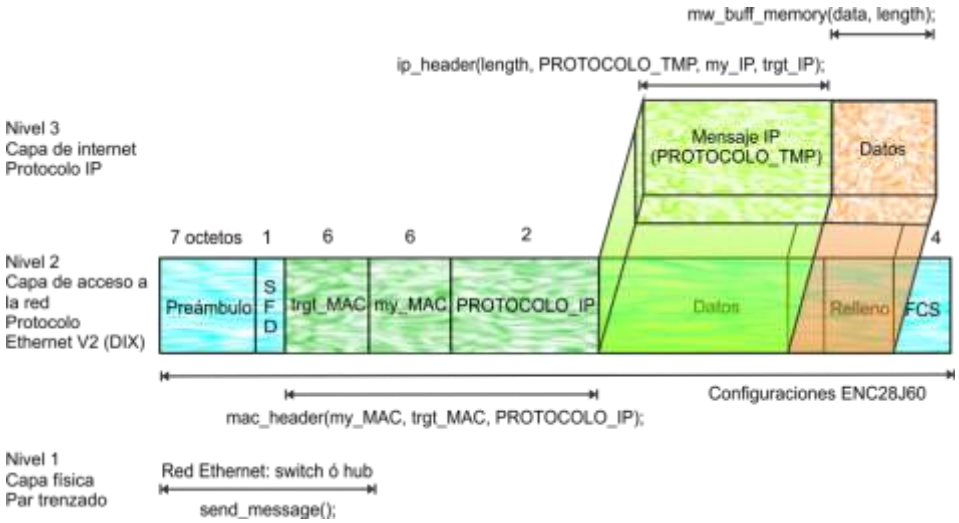


Figura 3.4.- Funcionamiento de envió de paquete IP a trasmitir por el sistema embebido.

int receive_package(char *data, char *my_MAC, char *my_IP, char *mask_network, char *trgt_MAC, char *trgt_IP).- Es una de las funciones principales de la librería y se encarga de procesar los paquetes Ethernet recibidos, tal como se muestra en el diagrama de flujo de la Figura 3.5.

Una de sus características más importantes es la de que dentro de la misma función se da respuesta a los mensajes ICMP y ARP de tipo petición. Donde también el valor que retorna es de suma importancia porque este le indica al usuario el tipo de paquete que se recibió, ya que:

- Si es 0, le indica que se contestó un mensaje ICMP de petición de eco, el cual es un evento que no es de su interés propiamente, en otro posible caso le indicaría que el mensaje se encontraba corrompido y por ello se descartó o en su defecto significaría que no había mensaje nuevo para procesar.

- Si es 1, le indica que se contestó o se recibió contestación a un mensaje ARP, como el protocolo solo cambia en su campo *operación,* para ambos tipos de mensaje se posee relacionada una dirección IP a una dirección MAC de un dispositivo con el cual se puede comunicar, siendo así se puede utilizar esta característica para generar una memoria de tipo caché y así lograr una comunicación más eficiente.

- Si es mayor a 1, le indica que se recibió un paquete con el identificador *PROTOCOLO_TMP*, así que tiene un vector de datos para procesar con longitud igual a este valor, de modo tal que el usuario debe saber el significado y organización de los mismos.

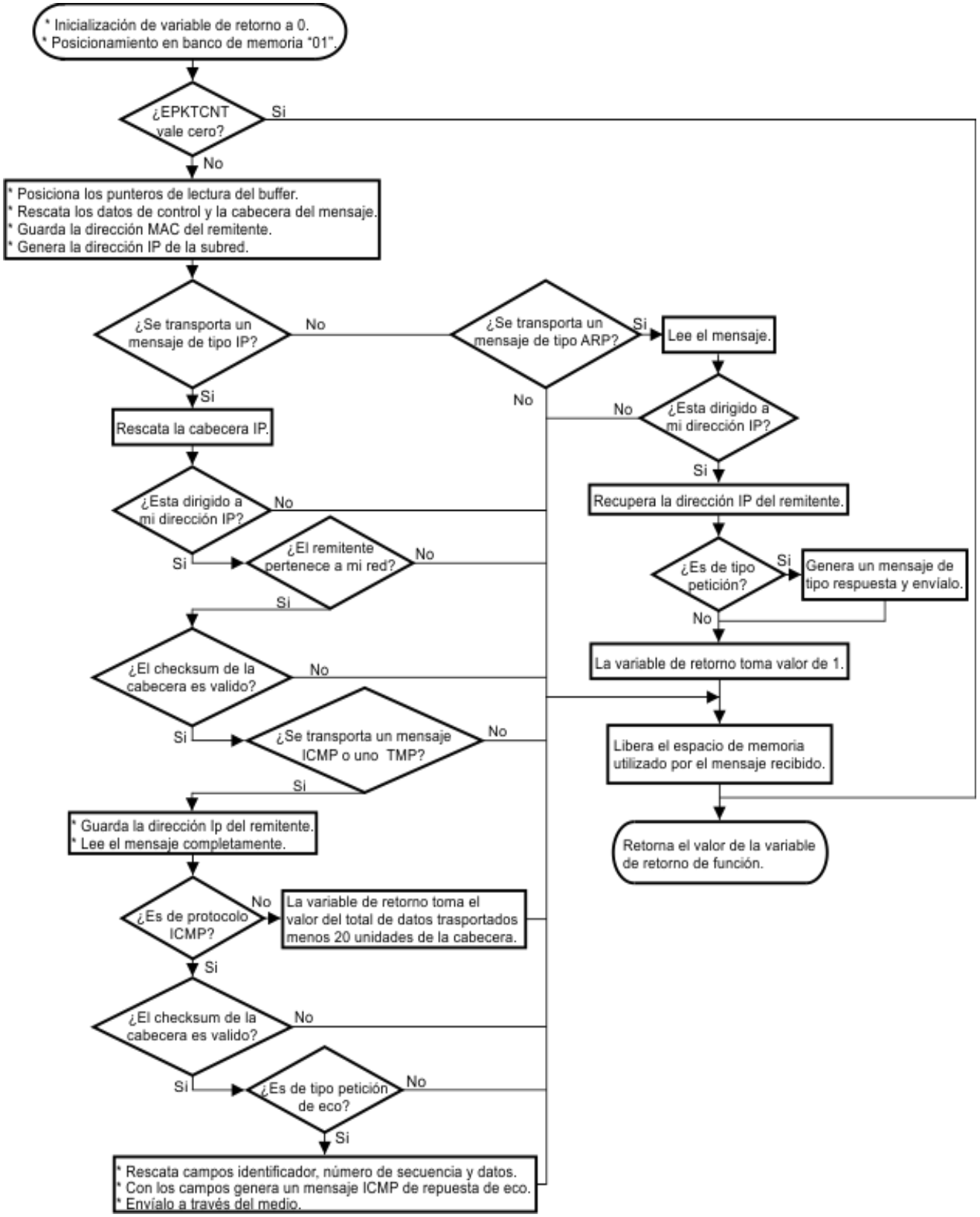


Figura 3.5.- Funcionamiento de función receive_package().

3.3 MANEJO DE SOFTWARE: MODBUS RTU Y ENHANCED SHOCKBURST™

Las librerías que implementan control sobre la comunicación serial RS-485 y sobre la comunicación bajo protocolo Enhanced ShockBurst™ fueron desarrolladas en proyectos previos al presente y su realización fue de tal forma que su uso fuese simple e intuitivo, donde algunas de sus características que se revisaron para poder utilizarlas se presentan a continuación:

Modbus RTU.- Para acceder a la comunicación es necesario hacer la inclusión del archivo de cabecera *RS485.h* en el archivo agregado al proyecto como archivo de compilación, al hacer esto automáticamente se tiene acceso a las funciones básicas para enviar y recibir mensajes a través del bus RS-485, y de este punto en adelante las mismas se deben manipular de tal modo que se emule el comportamiento del protocolo Modbus RTU (Figura 2.15):

rs485_bauds().- La frecuencia máxima a la cual la comunicación serial puede ser utilizada a través del sistema embebido es de 800 KHz, por lo tanto la función actual se encargada de configurar el pre-escalador de tal frecuencia.

rs485_push().- Escribe un byte de datos dentro del buffer de transmisión.

rs485_pop().- Se utiliza para leer un byte de datos del buffer de recepción.

rs485_flush().- Se encarga de limpiar ambos buffer.

rs485_send().- Cuando los paquetes Modbus se encuentran confeccionados esta función comienza su envió a través del bus.

rs485_wait_tx().- Cumple con la tarea de esperar a que el hardware envié completamente el mensaje a través del bus y así evitar que se intente realizar otro envió.

rs485_received().- Es necesaria para que el hardware se mantenga a la espera de respuesta, ya que su valor refleja si se ha recibido o no mensaje.

Enhanced ShockBurst™.- Se habilita su uso al incluir el archivo de encabezado *nrf2401.h* y algunas de las funciones básicas para lograr la comunicación son:

nrf_set_tx_addr().- Se encarga de configurar la dirección de 4 bytes que manejará el dispositivo cuando trabaje en modo transmisor.

nrf_set_rx0_addr().- Configura la dirección de 4 bytes que manejará el dispositivo cuando trabaje como receptor primario en la terminal de datos 0 y además también el tamaño del campo de datos del paquete que transmitirá.

nrf_init().- Es utilizada para llevar a cabo las configuraciones necesarias para que el dispositivo transmita y reciba paquetes dentro de la red inalámbrica. Recibe como argumentos el canal de frecuencia en el que operará y la terminal de datos que debe habilitar para que el dispositivo reciba los paquetes.

Todas las configuraciones anteriores deben ser iguales en un par de dispositivos, de modo que mientras uno se encuentre en modo de trasmisor el otro se encuentre como receptor.

nrf_send_data().- Es la encargada de enviar un mensaje a través del medio, para ello recibe los datos a ser enviados y la cantidad de los mismos.

nrf_check_int().- Es la encargada de checar si el pin de interrupción *IRQ* se encuentra en bajo, lo que enmarcaría que se recibió un paquete.

nrf_get_rx_pipe().- Necesaria para determinar cuál es la terminal de comunicación de datos activa.

nrf_get_data().- Recupera los datos recibidos por la terminal de datos.

3.4 REVISIÓN Y SELECCIÓN DE SENSORES DE LA IMU

La unidad de medición inercial que se encuentra integrada a la tarjeta *DUA-I* es manejada y controlada mediante un módulo conectado al sistema embebido que conjunta los manejadores de los dispositivos *L3G4200D* y *LSM303DLHC*. De manera general la IMU basa su funcionamiento en tecnología MEMS y se compone por:

- Un giroscopio de 3 ejes, el cual es un sistema que permite medir velocidades angulares.
- Un acelerómetro de 3 ejes, que permite realizar la medición de la aceleración gravitacional y de la aceleración que es provocada por movimiento.
- Un magnetómetro de 3 ejes, con el cuál se puede medir el campo magnético de la tierra y los campos generados por dispositivos eléctricos.
- Un sensor de temperatura embebido en el IC *LSM303DLHC*, que sirve para medir la temperatura interna del chip.

Para utilizar los sensores de la IMU en cuestiones de monitoreo se revisó que es necesario incluir el archivo de cabecera *imu.h*. En el proyecto se hizo uso del acelerómetro 3D y el magnetómetro 3D para poder poner bajo prueba los distintos enlaces de comunicación y así obtener parámetros de análisis. Se eligieron estos sensores con la finalidad de acotar el uso de sensores de la IMU y para acceder a ellos se utilizaron las siguientes funciones:

sync_read_accel().- Obtiene la medición forzada del acelerómetro en sus 3 ejes, y para lo cual entrega su valor en formato complemento a 2. Estos valores adicionalmente deben ser multiplicados por un factor de 1/16384 para obtenerlos en unidades *g*.

sync_read_mag().- Ejecuta la medición forzada del magnetómetro en sus 3 ejes, y en este caso las ganancia que se utilizan para obtener las mediciones en unidades *gauss (G)**,*** son de 1/1100 para los ejes X y Y, y de 1/980 para el eje Z.

De manera más específica para analizar las mediciones del magnetómetro fue necesario conocer que éste obtiene las componentes tridimensionales del campo magnético que se encuentre siendo medido sobre cualquier punto, es decir que se trata de un campo vectorial el cual posee tanto *dirección* como *magnitud*. El campo magnético o también llamado *densidad de flujo magnético (B),* es medido en unidades *gauss (G)* en el Sistema Cegesimal de Unidades y en unidades *tesla (T)* en el Sistema Internacional de Unidades.

Se conoció además que el campo magnético terrestre en magnitud oscila entre 0,25 y 0,65 G, donde la más común es la de alrededor de 0,5 G ya que su valor depende de las interferencias causadas por aparatos eléctricos así como de metales que se encuentren cercanos a donde se lleve a cabo la medición. También mediante las componentes vectoriales del campo magnético medido es posible conocer el ángulo de declinación respecto al norte magnético terrestre y así poder orientarse tal como si se usará una brújula. La orientación se conoce a través de la fórmula:

$$\theta = \begin{cases} 90 - \frac{180}{\pi}\operatorname{atan}\left(\frac{M_x}{M_y}\right) & M_y > 0 \\ 270 - \frac{180}{\pi}\operatorname{atan}\left(\frac{M_x}{M_y}\right) & M_y < 0 \\ 180 & M_y = 0, M_x < 0 \\ 0 & M_y = 0, M_x > 0 \end{cases} \qquad \text{(Ecu. 1)}$$

Por otro lado se conoció que el acelerómetro basa sus mediciones en unidades ***g,*** donde 1 *g* es la aceleración causada por la gravedad terrestre sobre cualquier objeto. Al ser un sensor tridimensional se puede medir aceleración en cualquiera de sus 3 ejes, donde el valor de las mediciones depende del posicionamiento del sensor, entre otros factores.

3.5 INTERFAZ GRÁFICA DE USUARIO

Para la programación de la interfaz gráfica de usuario se utilizó *CodeBlocks***,** el cual es un entorno de desarrollo bajo licencia GPL (General Public License, Licencia Pública General) y es utilizado para la creación de aplicaciones en lenguaje C y C++ mediante el compilador *Mingw/GCC,* también de acceso libre. Además del compilador, el entorno integra un editor orientado al lenguaje C y C++ con herramientas de autocompletado y resaltado de palabras clave, un depurador visual (debugger), un visor de proyectos y ficheros, y otras herramientas complementarias.

CodeBlocks permite a los usuarios acceder a diversas librerías que han sido desarrolladas con la finalidad de poder crear aplicaciones en consola y de igual manera para la creación de interfaces de usuario. En este último caso se hace de la API (Application Programming Interface, Interfaz de Programación de Aplicaciones) *gtkmm* de C++, la cual es un envoltorio de C++ para GTK+, una biblioteca utilizada para crear interfaces gráficas de usuario.

La programación de la interfaz se llevó a cabo mediante técnicas de programación convencionales en complemento con la revisión de la documentación y ejemplos proporcionados por desarrolladores del proyecto *gnome* en la página de internet https://developer.gnome.org/gtkmm-tutorial/stable/index.html.

La interfaz contó con el desarrollo de 4 principales secciones para la visualización y entrada de información al sistema, las cuales son resaltadas dentro de la Figura 3.6. Cada una de las secciones se programó para cumplir con un propósito:

Sección de visualización de variables.- En su mayoría se estructuró mediante el uso de widgets de tipo etiqueta con la finalidad de generar un espacio que presentará información de manera visual y así poder apreciar el cambio en las

mediciones al monitorear los 3 ejes tanto del acelerómetro como del magnetómetro. De igual manera en este espacio con ayuda de las mediciones del magnetómetro y teniendo en cuenta que la interfaz se adaptaría para el control de orientación de un sistema físico, se introdujo un widget de tipo entrada para así poder introducir la referencia en grados para el sistema de control, una barra de progreso para hacer visible que el sistema se encontraba en estado encendido o apagado (*on/off*) y una etiqueta para mostrar el posicionamiento actual del sistema y así tener una medida de comparación entre este valor y el de referencia, de cierta forma adaptando la interfaz a las características de un sistema SCADA.

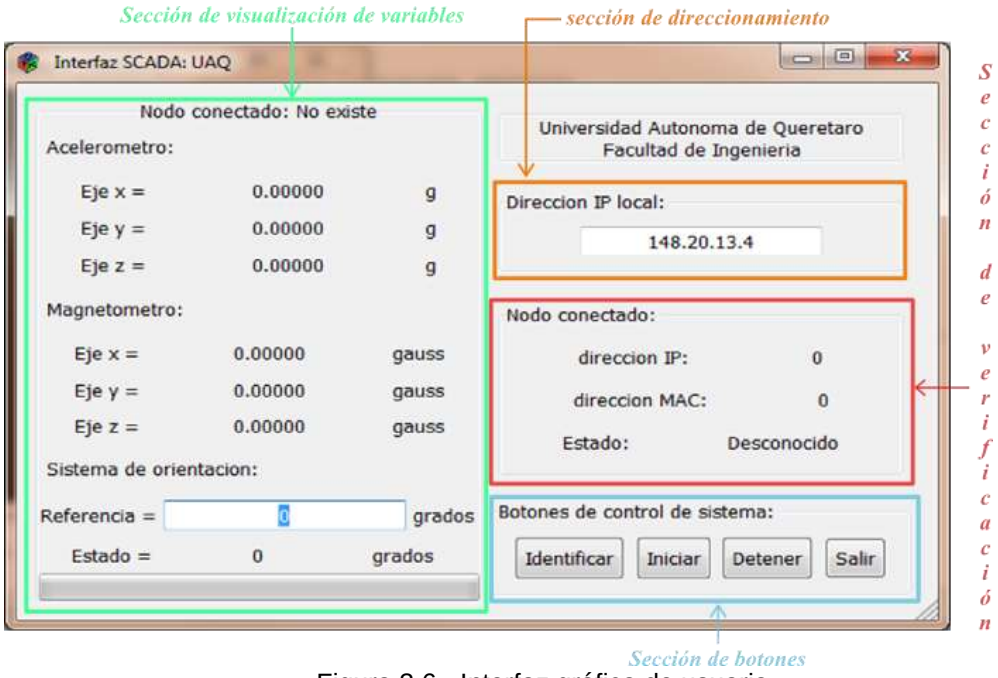


Figura 3.6.- Interfaz gráfica de usuario.

Se programó que al iniciar el sistema las etiquetas se actualizarán hasta que el sistema de control se detuviera por haber llegado a la referencia especificada, dando paso a poder introducir una nueva referencia y de nueva cuenta ejecutar todo el proceso anterior.

Sección de direccionamiento.- Se programó mediante el uso de un widget de tipo entrada, para que así el usuario introdujera la dirección IP del nodo con el cual se deseará comunicar dentro de la red Ethernet, es decir para que la interfaz

se pudiera comunicar con distintos nodos dentro de la red pero siempre sujetando la conexión de un solo nodo a la vez.

Sección de verificación.- Es de carácter informativa, ya que mediante etiquetas y después de haber presionado el botón “identificar” de la sección de botones, se programó para mostrar la conexión actual de la interfaz con el nodo, esperando que si la conexión fuese exitosa además de mostrar el valor de la dirección IP (el mismo que el de la entrada en la sección de direccionamiento), debería mostrar el valor de la dirección MAC que maneja el nodo a conectar, y en el apartado de estado marcaría la leyenda “Activo”.

Sección de botones.- Aquí se reúnen los botones que controlan la secuencia de funcionamiento de la interfaz y de manera directa el de la red de comunicaciones, se comienza con el botón “Identificar” el cual al ser presionado se programó para obtener la dirección IP del nodo a conectar y enviarla junto con un comando codificado al sistema embebido mediante comunicación USB y así almacenar la dirección internamente, después debía enviar otro mensaje codificado requiriendo que el sistema embebido hiciera la petición de la dirección MAC que utiliza el nodo dentro de la red Ethernet y que lo retroalimentará con tal información, por último debía enviar de nueva cuenta la dirección MAC codificada con la intención de almacenarla de manera interna tal como con la dirección IP. El botón “Iniciar” se programó para que al detectar su señal de activación o “clicked” creara un “hilo de multiprocesamiento” (thread) con duración indefinida en el cual constantemente se enviará a través de USB el comando establecido para la acción de control y monitoreo del acelerómetro y magnetómetro, siempre y cuando el sistema no llegará a la referencia agregando que en este mismo “hilo” se realizaría el procesamiento correspondiente a cada medición de los sensores y en base a esto se determinaría el valor de la señal de control que sería enviada en el siguiente mensaje, de igual forma se programó que si los datos procedentes de la red enmarcaban que al recibirlos no existió problema dentro de la red, se ejecutaría un segundo “hilo” con el cual se actualizarían los valores de las

etiquetas. El botón “Detener” se programó de modo que si en determinado momento y con el sistema corriendo “hilos” se pudiese detener el proceso, pero de manera principal se creó con la finalidad de poder enviar señales de control de apagado a través de la red y así iniciar nuevas acciones de manera segura. El botón “Salir” se programó para que además de ejecutar las mismas acciones que el botón “Detener”, terminará la ejecución de la interfaz.

3.6 IMPLEMENTACIÓN DE RED MULTIPROTOCOLO

Al conocer que mediante las mediciones de las componentes del campo magnético terrestre es posible orientarse y además por el hecho de poder acceder a estas a través del magnetómetro, se propuso la creación de un sistema de orientación de esta naturaleza, el cual se muestra junto con su mecanismo electrónico de accionamiento en la Figura 3.7.

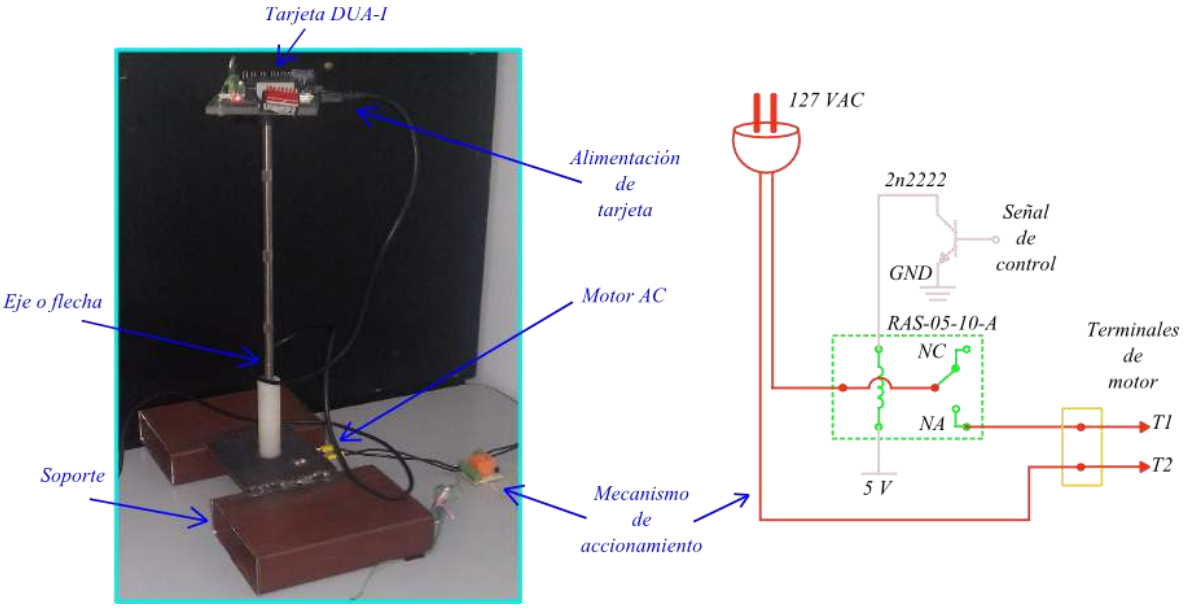


Figura 3.7.- Sistema físico y mecanismo de accionamiento.

El sistema básicamente se encuentra conformado por un *soporte* sobre el cual se monta un *motor* con caja de engranes y alimentación a 127 *VAC,* cuenta con un *eje* montado al rotor que funciona como extensión para que de esta forma

los campos magnéticos generados dentro del motor no afecten las mediciones del magnetómetro, por ultimo al termino del eje y sobre una base plástica se encuentra montada una tarjeta *DUA-I*.

En lo que respecta al mecanismo de accionamiento se hace uso de un relevador de estado sólido, donde debido al valor en voltaje de la *señal de control on/off,* se realiza el cambio de estado del contacto normalmente abierto (*NA*) al energizarse la bobina, lo que permite o impide el paso de voltaje de alimentación hacia el motor.

Por otra parte en cuanto a la red de comunicaciones y tal como se especificó en el planteamiento metodológico, para comprobar de manera contundente que los distintos protocolos pueden interactuar entre sí, se debe estructurar la red de tal manera que se haga uso de los 3 protocolos principales para la comunicación y envío de señales de control, siendo así se basó en la Figura 3.1 para estructurar la implementación de la red tal como se muestra en la Figura 3.8.

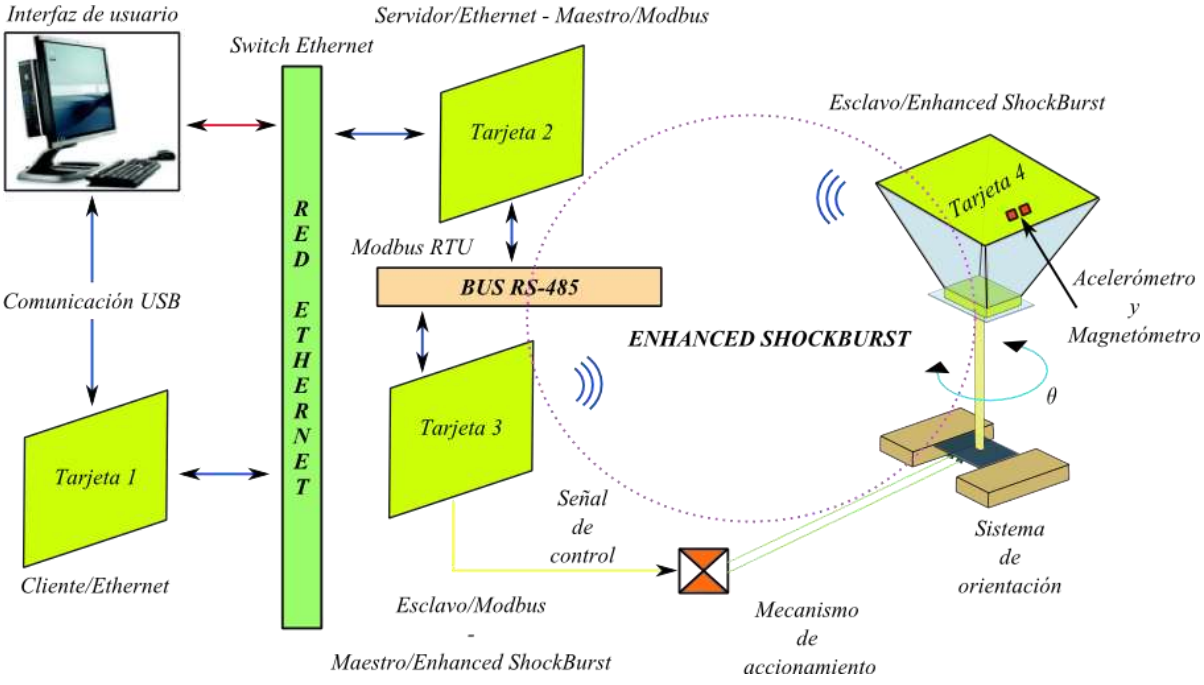


Figura 3.8.- Implementación de red.

Para cumplir con el funcionamiento de la red de comunicaciones fue necesario desarrollar software para cada una de las tarjetas utilizadas dentro de la red. El software puede ser revisado en el apartado de apéndice correspondiente a partir de la sección 6.2 y su descripción general se muestra a continuación:

- Tarjeta 1 (Cliente/Ethernet).- Se programó para desempeñarse dentro de la red Ethernet como *Cliente,* ya que sería el dispositivo que requeriría que los nodos adicionales dentro de la red le entregarán información, pero además sería el vínculo directo con la interfaz gráfica de usuario a través de USB. La programación consistió en realizar las configuraciones necesarias para comunicarse bajo ambos protocolos, después el resto del procesamiento recaía en la interrupción generada por USB, en la cual se decodificaba el mensaje para conocer cuál de 5 posibles acciones programadas en el dispositivo sería ejecutada, así como también conocer el valor de la señal de control (0 o 1), entre otros. Las acciones programadas fueron:
    1. Almacenar IP.- Al ejecutarse se decodificaba del mensaje USB la dirección IP que sería almacenada dentro de variables estáticas y así poder manipular esta dirección internamente.
    2. Identificar.- Necesaria para conocer la dirección MAC que maneja el dispositivo que posee la dirección IP almacenada, en ella antes que nada se corrobora que la dirección pertenezca a un dispositivo dentro de la red, siendo así esta dirección sería asignada al vector correspondiente para la dirección del nodo IP a conectar, si no fuese así se asignaría a este vector la IP del router y así cumplir con el mecanismo de encaminamiento a través de redes. También se envía un mensaje de tipo ARP, esperando una posible respuesta dentro de un límite de tiempo (“time out”), con la dirección MAC obtenida se genera un mensaje USB de respuesta.
    3. Almacenar MAC.- Se recibe la dirección MAC que anteriormente se envió como respuesta para ser almacenada tal como con la IP.

4. Control y monitoreo.- En esta acción se acceden a los valores de las direcciones MAC e IP almacenadas para poder dirigir paquetes IP con información específica a determinado nodo dentro de la red. Se espera respuesta en un tiempo límite de *12 ms* (máximo esperado para que los datos atraviesen toda la red) con la información obtenida se generan mensajes de respuesta, enmarcando en cada caso si se presentó problemas de tiempo límite.
5. Detener sistema.- En este caso solo se le envía un mensaje IP con el comando de *detener el sistema* al nodo conectado.

- Tarjeta 2 (Servidor/Ethernet – Maestro/Modbus).- Se programó como *servidor* dentro de la red Ethernet para que solo cumpliera con la tarea de proporcionar al *Cliente* la información que requiriera, pero por otro lado fungía como maestro dentro del bus RS-485 haciendo uso del protocolo Modbus. Se configuró con una dirección IP y MAC de modo tal que perteneciera a la red de trabajo y se comenzó por configurar timers, interrupciones, al dispositivo ENC28J60 y la frecuencia de transmisión a través de RS-485, después solo se programó para que estuviera a la expectativa de paquetes dirigidos a él y que aquellos mensajes que al ser leídos retornarán un valor mayor a 1 (protocolo TMP) fuesen procesados para definir qué acción ejecutar, siendo 2 posibles casos:
    1. Control y monitoreo.- Aquí el dispositivo se comportaba como pasarela entre protocolos, tomaba la información del mensaje Ethernet recibido y confeccionaba un mensaje bajo protocolo Modbus, lo enviaba y esperaba por una respuesta que no estuviera fuera del rango del tiempo límite (si ocurría este suceso lo informaba a través de Ethernet), al obtener repuesta a través del bus tomaba la información y confeccionaba un mensaje Ethernet de repuesta.
    2. Detener sistema.- Solo hacia el cambio de información del mensaje ethernet recibido al mensaje Modbus a ser enviado, para lo cual no

esperaba repuesta y tampoco generaba una para ser enviada a través de Ethernet.

- Tarjeta 3 (Esclavo/Modbus – Maestro/Enhanced ShockBurst).- Se determinó que esta tarjeta fungiría como esclavo dentro del bus RS-485, maestro al utilizar el protocolo Enhanced Shockburst™ y que además en este punto la señal de control tendría validez, es decir que está tarjeta proporcionaría una de sus salidas digitales para hacer posible el envío físico de la señal de control hacia el mecanismo de accionamiento. Se configuró la frecuencia de transmisión a través de RS-485 y de igual manera las configuraciones básicas de la comunicación inalámbrica, después se mantenía a la espera de la llegada de mensaje a través del bus RS-485, cuando recibía mensaje lo procesaba para conocer las acciones que eran requeridas, si se trataba de *control y monitoreo* de manera parecida a la tarjeta 2 actuaba como pasarela, solo que ahora el cambio era de protocolo Modbus RTU a protocolo Enhanced Shockburst™ y viceversa al recibir respuesta a su mensaje, donde además se mapeaba la señal de control en el bit menos significativo de la sección de salidas digitales, pero en cambio si se trataba de *detener el sistema* simplemente se restablecía a cero el bit de las salidas digitales utilizado.
- Tarjeta 4 (Esclavo/Enhanced ShockBurst).- Solo fue programada para que ejecutará y retornará las mediciones del acelerómetro y magnetómetro al recibir un mensaje a través de comunicación inalámbrica.

De manera general para que la información correspondiera al confeccionar cada paquete, se trató de seguir el mismo método de ordenamiento y los mismos valores para funciones similares pero desempeñadas por diferentes tarjetas, agregando que hasta la 3er tarjeta se contó con un mecanismo para determinar el sobrepaso de tiempo límite en la comunicación y así con ello evitar que las mismas se quedarán de manera indefinida en un estado de espera hasta recibir respuesta.

4.1 METODOLOGÍA DE PRUEBAS

En este punto al contar con el software para la transferencia de datos a través de Ethernet y además conocer el manejo de los distintos protocolos de comunicación involucrados, fue necesario poner bajo prueba su funcionamiento mediante la creación de pequeños sistemas en los cuales pudieran ser obtenidos y analizados parámetros importantes en torno a la transferencia de mediciones de sensores, tales como: tasas de transferencia y frecuencias de muestreo. Los sistemas que fueron sujetos de análisis pueden ser visualizados mediante la Figura 4.1, donde se observa que las distintas redes de comunicación se analizan de manera individual y combinada.

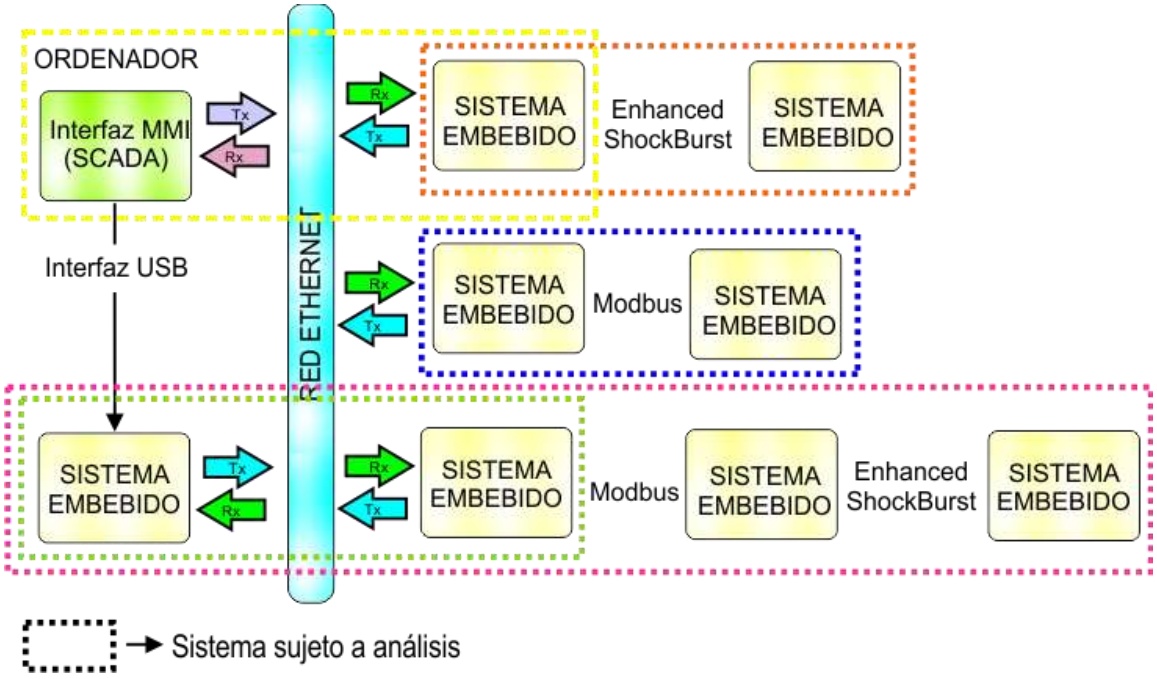


Figura 4.1.- Metodología de pruebas de comunicación.

De igual forma las conexiones que se llevaron a cabo mediante cableado entre la tarjeta *DUA-I* y la tarjeta *ENC28J60-H*, y el módulo transceptor *nRF24L01+*, se pueden encontrar en la Tabla 4-1.

Tabla 4-1.- Conexión de tarjetas.

Puerto de expansión *DUA-I*			***ENC28J60-H***		***Módulo nRF24L01+***	
Señal	Terminal	Pin	Señal	Pin	Señal	Pin
CS	L17	13	CS	7	CSN	6
SCK	H15	14	SCK	1	SCK	5
SO	J15	15	MOSI	2	MOSI	4
SI	H16	16	MISO	3	MISO	3
INT	K14	17	INT	5	IRQ	2
WOL	K15	18	WOL	4	---	---
CE	M16	19	RST	8	CE	7
GND	---	1	GND	9	GND	1
3.3V	---	3	3.3V	10	VCC	8

4.2 ANÁLISIS DE COMUNICACIÓN MEDIANTE PROTOCOLO ETHERNET

De primera cuenta se analizó el sistema en el cual solo se conectaron dos dispositivos manejados por el sistema embebido a la red Ethernet, uno de ellos funcionando dentro de la red como *cliente* y el otro como *servidor*. Para la conexión de ambos fue necesaria la creación de la red física mediante el uso de un *switch Ethernet* en conjunto con un par de cables de *par trenzado*.

En lo que respecta a la programación del servidor, solo se realizaron las respectivas configuraciones del dispositivo (dirección IP, MAC, etc.), se inicializó para que recibiera paquetes Ethernet y se programó de modo que aquellos paquetes que en su cabecera IP contuvieran el identificador *PROTOCOLO_TMP* los retornará como tal al cliente. Para lograr contar el tiempo transcurrido desde la petición hasta la obtención de datos se hizo uso de la librería *timers.h* ya que permite acceder a herramientas de temporizado. El cliente además de sus configuraciones de dispositivo e inicialización, configuró el temporizador a utilizarse y fue programado para que antes de enviar un paquete con 1200 bytes de información útil, habilitara el conteo del temporizador, lo detuviera justo después de recibir el mensaje de respuesta, almacenara su conteo y que lo limpiara para poder ejecutar uno nuevo.

En base al funcionamiento anterior se realizaron 100 muestras de este proceso y los tiempos obtenidos se muestran en la gráfica de la Figura 4.2, donde se puede observar que las muestras se mantienen en un valor casi constante ya que presentan un grado de dispersión casi nulo, lo que enmarca que la comunicación a través de Ethernet fue sumamente estable.

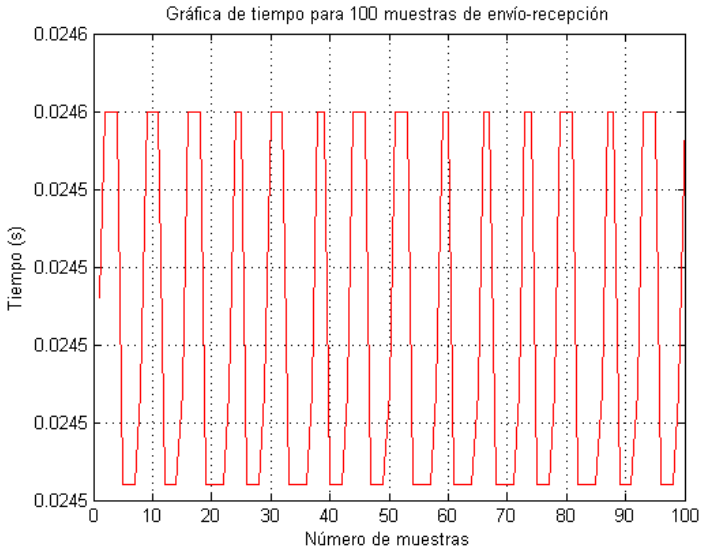


Figura 4.2.- Gráfica de muestreo del proceso envío-recepción de 1200 bytes a través de Ethernet.

Con los datos referentes a las muestras se obtuvo un tiempo promedio de ***0,0245 s*** para la ejecución del proceso, así por lo tanto la tasa de transferencia de datos de carga útil pudo ser calculada como:

$$\boldsymbol{tasa\ de\ transferencia} = \frac{2(1200)(8\ bits)}{0.0245\ s} \div 2^{20} \approx \mathbf{0.75}\ \boldsymbol{Mbps}$$

4.2.1 PRUEBA DE GESTIONAMIENTO DE RED

Para corroborar que la transmisión de datos se llevará de la manera correcta a nivel de capa de internet, también conocido como nivel IP, se hizo uso de la herramienta *ping*, la cual sirve para diagnosticar la conectividad de red entre dispositivos ya que implementa mensajes ICMP de varios tipos que a su vez viajan dentro de un datagrama IP. El comando ping en computadores bajo sistema

operativo Windows® puede ser ejecutado a través de la ventana de comandos del programa *símbolo del sistema o cmd*. En este caso se utilizó para la petición de mensajes eco, donde previamente se requirió llevar a cabo la configuración de una red de área local por parte del ordenador***.***

La ejecución del comando y los datos de la comunicación establecida con el sistema embebido pueden ser observados en la Figura 4.3, donde además de enmarcar que no se presentó perdida de paquetes, se proporcionó un tiempo promedio de ***3 ms*** para la transacción y con ello fue posible calcular una tasa de transferencia promedio para el proceso de:

$$\boldsymbol{tasa\ de\ transferencia} = \frac{2(32)(8\ bits)}{0.003\ s} \div 2^{10} \approx \boldsymbol{167\ Kbps}$$

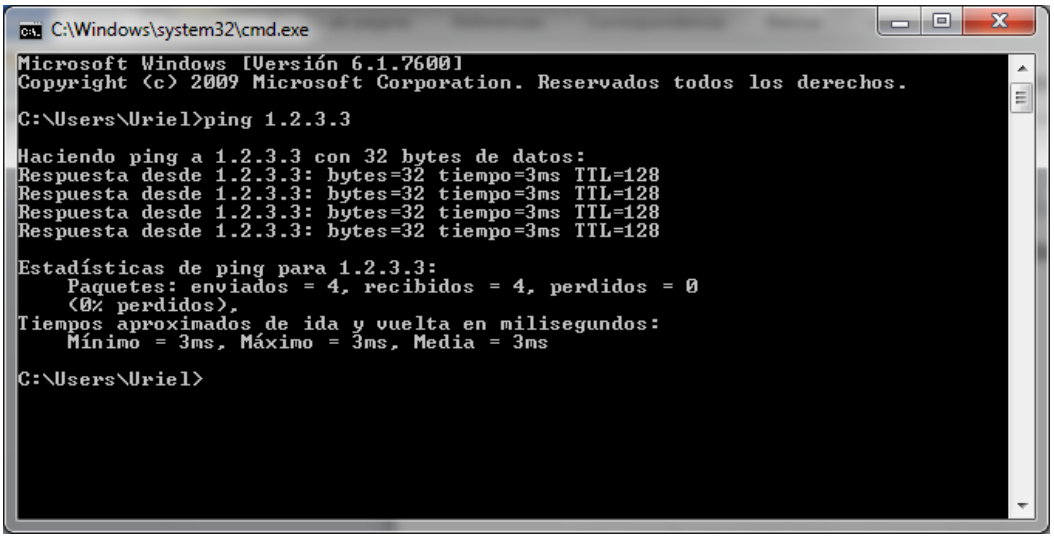


Figura 4.3.- Prueba de conectividad mediante herramienta *ping.*

La tasa obtenida difiere de la velocidad promedio de 256 Kbps que obtuvo Lugo (2013), al realizar la misma prueba y en cuyo caso fue mayor a la presente, pero es necesario considerar que durante la prueba también entran en juego las características propias del computador en el que se lleva a cabo, por ejemplo; la velocidad del procesador, los programas adicionales que se encuentren siendo ejecutados, entre otros, aun así como punto principal se comprobó que el software desarrollado comunica al sistema embebido con el computador de manera

correcta a nivel IP y que de igual manera lo hace la implementación de mensajes bajo protocolo ARP.

4.3 ANÁLISIS DE RED MODBUS RTU

Modbus RTU como ya se especificó es un protocolo de nivel de aplicación que en este caso deja las tareas de enlace a la comunicación serial RS-485. En base a esto la comunicación a través de Modbus RTU se analizó para un sistema que desempeñaba comunicación entre un dispositivo maestro y un dispositivo esclavo conectado al bus RS-485.

Ambos dispositivos fueron configurados para trabajar a la frecuencia máxima de 800 KHz. El dispositivo esclavo se programó de modo que solo contestará los mensajes recibidos siempre y cuando el primer byte correspondiera a su dirección de esclavo, donde además su respuesta dependía del segundo byte del mensaje, el cual es el correspondiente al campo *función* del protocolo Modbus RTU y en base a su valor respondía con las mediciones del acelerómetro o del magnetómetro según el caso.

El dispositivo maestro en cambio se programó para la petición de 1000 muestras del acelerómetro, de 100 muestras del magnetómetro midiendo la intensidad del campo magnético terrestre dentro de una aula de clases y además también para contar el tiempo de cada transacción, obteniéndose un tiempo promedio de ***380,43 µs***, así por lo tanto se calculó que cada sensor en sus 3 ejes puede ser muestreado a una frecuencia de:

$$\boldsymbol{f_s} = \frac{1}{T} = \frac{1}{380.43x10^{-6}s} \approx \boldsymbol{2.628\ KHz}$$

Las mediciones del magnetómetro después de ser procesadas se muestran en la Figura 4.4, donde se aprecia que la magnitud del campo magnético terrestre es cercana a los 0,6 G y que al momento de medir la orientación del eje de medición ***x*** se encontraba a 304,9° respecto al norte magnético terrestre.

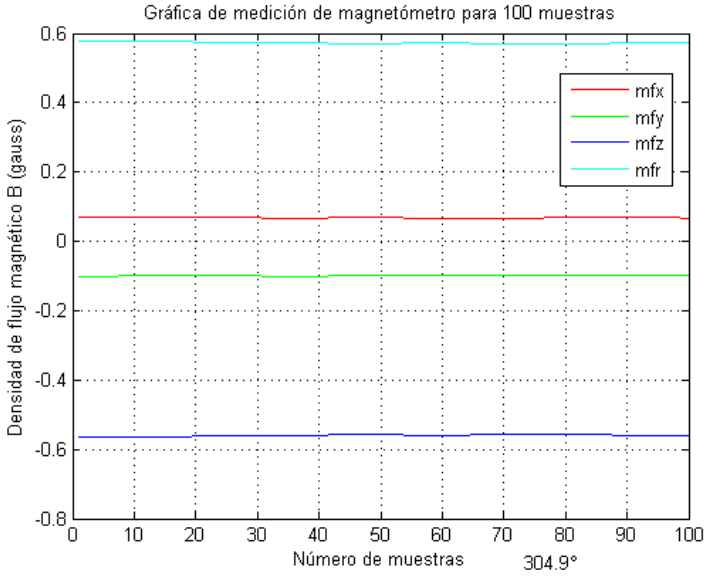


Figura 4.4.- Mediciones de magnetómetro obtenidas mediante protocolo Modbus.

En lo que respecta a las mediciones del acelerómetro, durante la prueba la tarjeta DUA-I descansaba de tal forma que el eje *z* del sensor estuviera bajo la acción directa de la gravedad, es decir que este eje debía presentar mediciones de alrededor de 1 *g*, en este caso para poder apreciar cambios en las mediciones se aplicó un leve golpe cercano a la tarjeta que repercutió en la medición de los ejes del sensor tal como se muestra en la Figura 4.5.

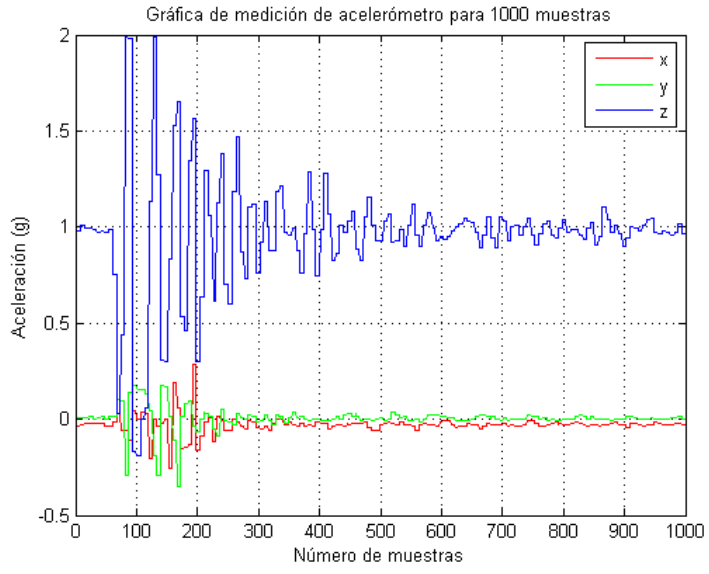


Figura 4.5.- Mediciones de acelerómetro obtenidas mediante protocolo Modbus.

Es importante notar en la Figura 4.5 que las mediciones parecen mantenerse de manera constante durante aproximadamente 6 o 7 muestras consecutivas, esto debido a que el sensor es forzado a realizar una medición y para lo cual los registros donde recae su valor no se actualizan tan rápido como se hacen las peticiones, por lo tanto encontrando en ellos el valor de mediciones pasadas, lo que enmarca la alta velocidad que maneja la comunicación serial y que podría ser una característica muy útil en ciertas aplicaciones.

4.4 ANÁLISIS DE PROTOCOLO ENHANCED SHOCKBURST™

Al igual que para el protocolo Modbus RTU está prueba se realizó para la obtención directa de las mediciones del acelerómetro y el magnetómetro pero en este caso a través de comunicación inalámbrica. En ella se programó un dispositivo para que dependiendo el valor del primer byte del mensaje que recibía, devolviera ya fuese la medición de uno u otro sensor. En caso contrario, tal como en las pruebas pasadas otro dispositivo fue programado para pedir 500 muestras del acelerómetro y 100 del magnetómetro, ambos en sus 3 ejes, y además también para encargarse de medir el tiempo para la transacción.

La petición de muestras consistía en un ciclo continúo desde el momento que comenzaba hasta finalizar y al ejecutarse se encontró con la dificultad de no poder establecer comunicación. Se resolvió el problema al agregar 800 µs de retraso entre cada muestra con el fin de proporcionar el tiempo suficiente para que el 2do dispositivo respondiera el mensaje. El tiempo de retraso fue la cantidad mínima que pudo ser utilizada y repercutió directamente en el tiempo de muestreo.

El tiempo promedio obtenido para cada muestra fue de ***1130 µs***, así por lo tanto la frecuencia de muestreo resulta de:

$$\boldsymbol{f_s} = \frac{1}{0.00113\ s} \approx \mathbf{884}\ \boldsymbol{Hz}$$

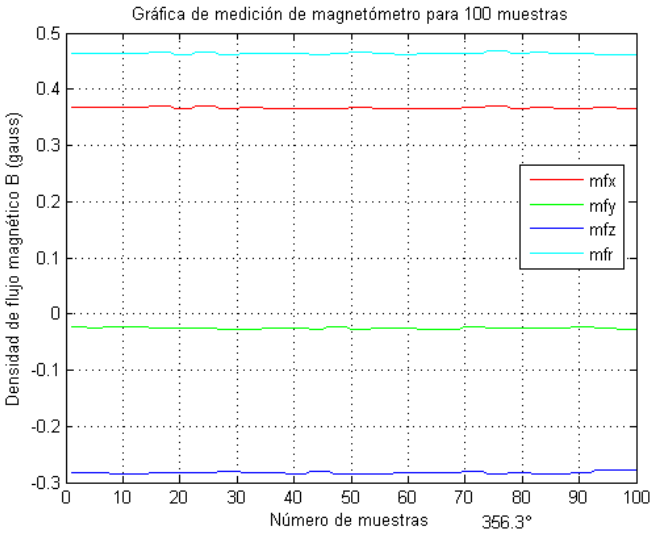


Figura 4.6.- Mediciones de magnetómetro obtenidas mediante comunicación inalámbrica.

En la Figura 4.6 se muestran las mediciones del magnetómetro referidas a cada componente tridimensional del campo magnético terrestre, donde de igual manera se presenta el campo magnético resultante (color cyan) y la orientación de 356.3° respecto al norte magnético terrestre.

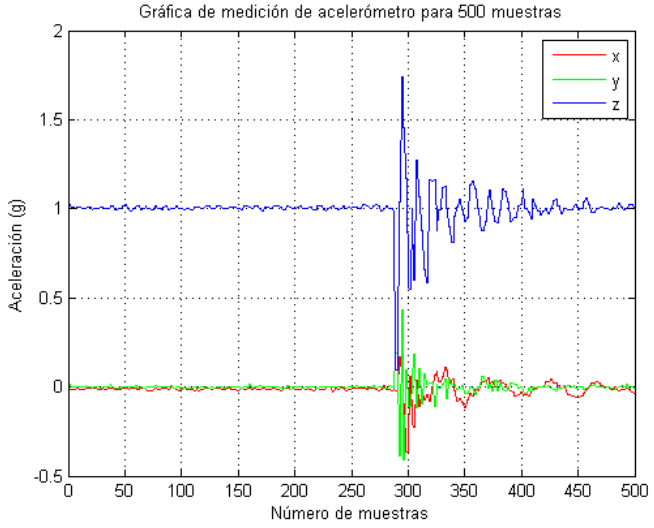


Figura 4.7.- Mediciones de acelerómetro obtenidas mediante comunicación inalámbrica.

Por otra parte en la Figura 4.7 se muestran las mediciones del acelerómetro obtenidas mediante el mismo método que para las pruebas con el protocolo Modbus RTU.

4.5 FUNCIONALIDAD DE PROTOCOLOS EN CONJUNTO

Otro análisis de sistema de pruebas de gran importancia para el proyecto fue en el cual se conjuntaban los tres diferentes protocolos manejados hasta el momento. Para la prueba y análisis se configuraron 4 dispositivos de la siguiente manera:

- Dispositivo 1.- Se programó para conectarse a la red Ethernet fungiendo como *cliente,* realizaba antes que nada sus configuraciones respectivas para comunicarse dentro de la red y seguido de ello enviaba un mensaje ARP de tipo petición al dispositivo que actuaba como *servidor* y con el cual se quería establecer la comunicación. En general mediante el envío de mensajes IP le requería al dispositivo servidor que se comunicara con determinado esclavo del bus RS-485, para lo cual en el primer byte le especificaba la dirección de esclavo y en el segundo el campo función. También se programó para esperar de manera indefinida hasta que recibiera contestación al mensaje enviado y para almacenar el tiempo utilizado para este proceso.
- Dispositivo 2.- Fungía como *servidor* dentro de la red Ethernet pero por otra parte debía comportarse como dispositivo maestro dentro del bus RS-485. Su labor consistía en recuperar los dos primeros bytes del mensaje IP recibido y con ellos confeccionar un mensaje bajo protocolo Modbus, hacia su envío y esperaba hasta que hubiese contestación por parte del dispositivo esclavo, con la información generaba un mensaje IP y lo enviaba como respuesta al dispositivo cliente.
- Dispositivo 3.- De igual manera este dispositivo tenía dos principales funciones, la primera era comportarse como esclavo dentro del bus RS-485 y la segunda comportarse como dispositivo maestro dentro de la red inalámbrica. Se mantenía a la escucha dentro del bus RS-485 hasta que recibiera mensaje, lo recuperaba y si el primer byte correspondía a su dirección de esclavo generaba un mensaje donde incluía el segundo byte

recibido y lo enviaba a través de la red inalámbrica, agregaba el tiempo respectivo de retraso y se colocaba a la espera de respuesta, con la información recibida generaba un mensaje de repuesta y lo enviaba al dispositivo maestro a través del bus RS-485.

- Dispositivo 4.- Este fue el dispositivo más alejado en el sistema, es donde terminaba la interconexión de protocolos y fue programado para ser dispositivo esclavo dentro de la red inalámbrica, de acuerdo al mensaje recibido contestaba al maestro con las mediciones del acelerómetro o el magnetómetro.

Ya programados los dispositivos y conectados mediante los respectivos medios a las diferentes redes y bus dentro del sistema, el dispositivo 1 se encargó de pedir 400 muestras del acelerómetro del dispositivo 4. Las mediciones se presentan en la gráfica de la Figura 4.8 donde además es posible notar que no se presenta el problema descrito en la prueba mediante Modbus debido a que en este caso se consume más tiempo al recorrer toda la red del sistema.

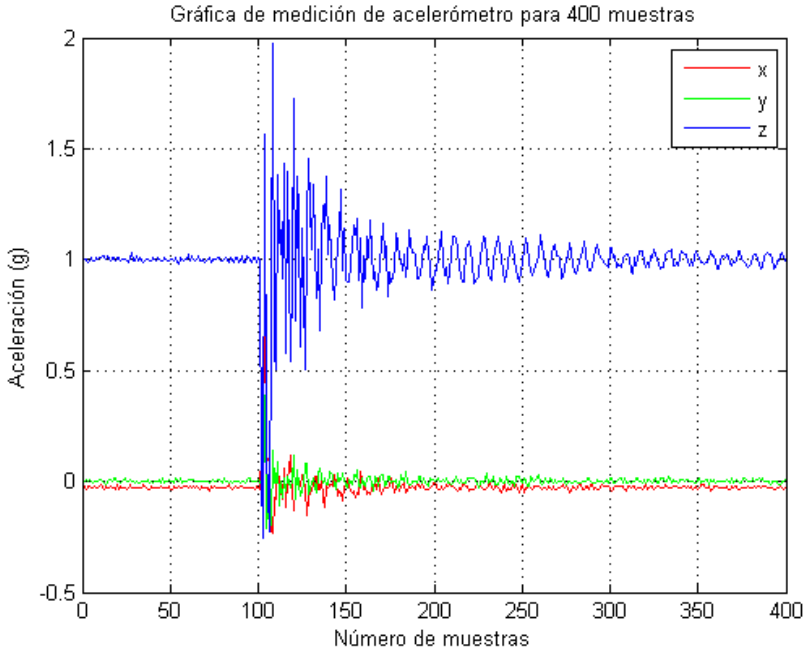


Figura 4.8.- Mediciones de acelerómetro obtenidas al conjuntar protocolos.

El tiempo promedio para que las muestras a travesarán toda la red fue de **3,736 ms** con lo cual fue posible calcular una frecuencia de muestreo de:

$$f_s = \frac{1}{3.736x10^{-3}\ s} \approx \mathbf{268\ Hz}$$

4.6 CONTROL Y MONITOREO A TRAVES DE LA RED

Después de haber analizado todos los sistemas de prueba propuestos y con ello habiendo obtenido parámetros de rendimiento para cada protocolo de comunicación, se culminó el desarrollo del proyecto al poner bajo prueba el funcionamiento del sistema completo, es decir conjuntar la interfaz gráfica, la red de comunicaciones y el proceso a ser sujeto de control y monitoreo, tal como se muestra en la Figura 4.9.

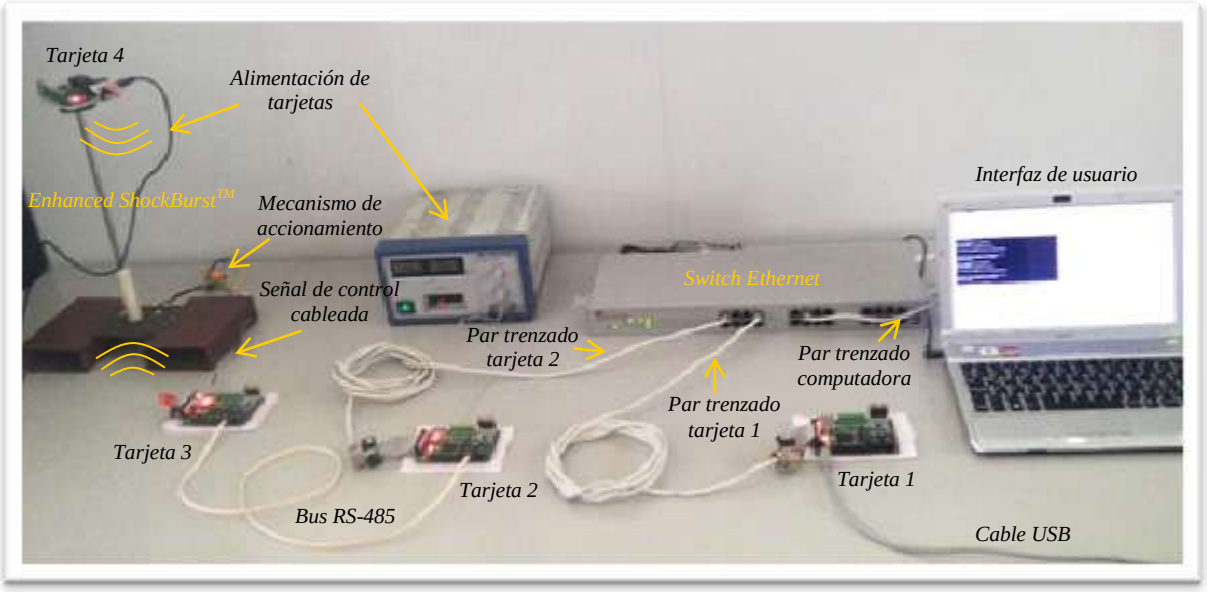


Figura 4.9.- Implementación de red de control y monitoreo.

Se comenzó por la programación de las 4 tarjetas con su correspondiente código fuente (*descripción en la página 82)* para después proseguir con las conexiones físicas dentro de la red, tal como: alimentación de tarjetas, cableado

de red, bus, etc. En este punto se poseía todo lo necesario para iniciar el sistema, así que se ejecutó la interfaz de usuario y se presionó el botón “Identificar” con el propósito de que se llevara a cabo la identificación del nodo que poseyera la dirección IP “148.20.13.4”, generando así cambios en la interfaz de la manera que se muestra en la Figura 4.10, donde en la sección de verificación además de mostrar la dirección MAC del dispositivo también se muestra que se encuentra en un estado activo.

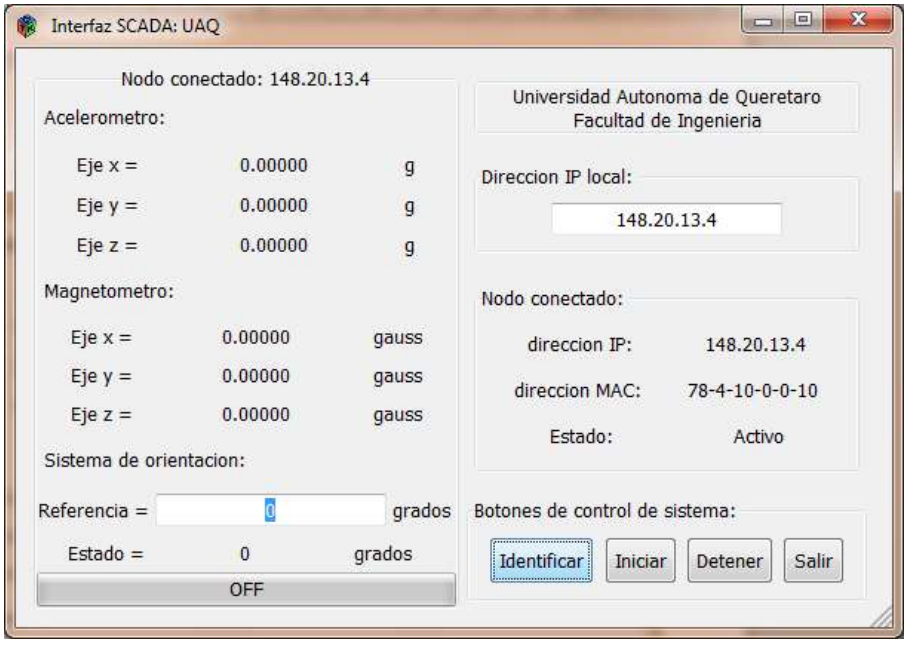


Figura 4.10.- Interfaz de usuario después de identificar nodo.

Ya establecida la conexión inicial, se continuó al introducir una referencia de 65° para el sistema de orientación y se presionó el botón “Iniciar”, así comenzó la comunicación continúa entre la interfaz y la red. Los cambios en las mediciones de los sensores pueden ser observados en la Figura 4.11, mientras que el fin de la comunicación por haber llegado a la referencia se muestra en la Figura 4.12.

Después de haber llegado a la referencia el sistema permitía introducir un nuevo valor y así comenzar de nueva cuenta el proceso, pero además también como segunda opción permitía generar una nueva conexión con algún otro nodo.

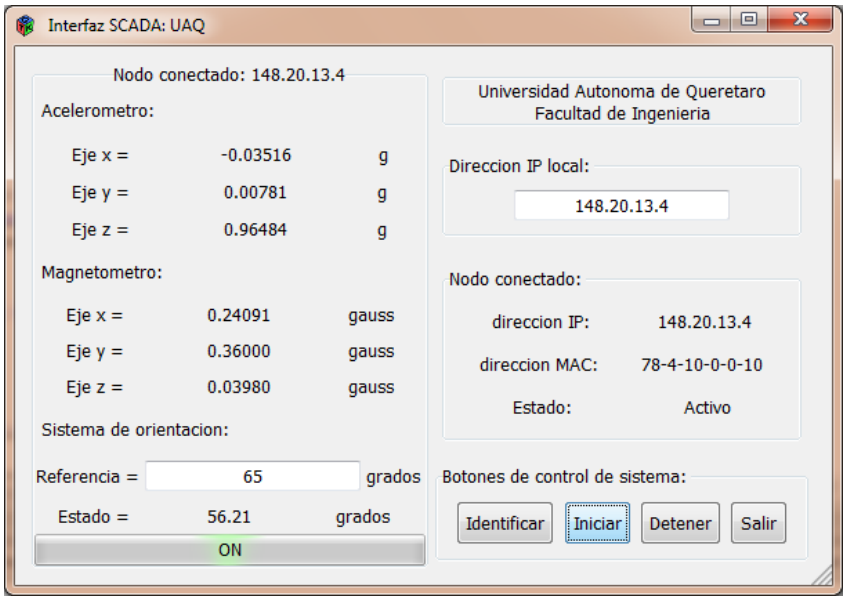


Figura 4.11.- Interfaz de usuario después de iniciar proceso de control y monitoreo.

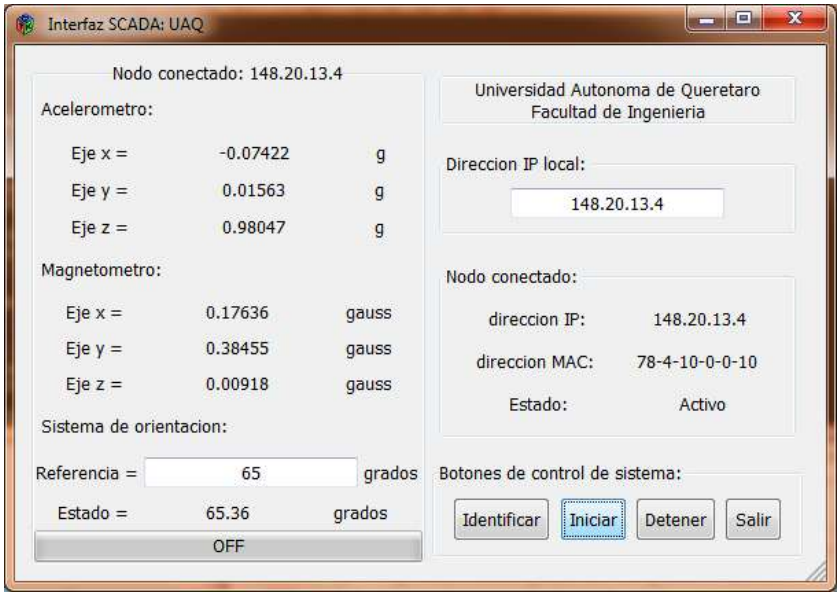


Figura 4.12.- Interfaz de usuario después de haber alcanzado la referencia.

Cabe mencionar que para corroborar que el sistema de orientación funcionará correctamente se comparó su orientación contra la de una aplicación de teléfono celular que emula el comportamiento de una brújula y para lo cual se encontraron ligeras discrepancias pero no tan significativas (2 o 3 grados de diferencia) ya que como tal las mediciones dependen del grado de sensibilidad del sensor magnetómetro utilizado.

5. CONCLUSIONES Y PROSPECTIVAS

5.1 CONCLUSIONES

El desarrollo del proyecto permite apreciar claramente que en la actualidad tanto el uso de sensores como el uso de comunicaciones son aspectos de gran relevancia que impactan no solo el ámbito industrial y el de investigación, sino que además también estas herramientas tecnológicas se han ajustado para tener cabida y cumplir con los requerimientos de algunos otros ámbitos diferentes.

En un mundo tan competitivo como el de hoy en día, el desarrollo de herramientas cada vez más innovadoras y versátiles es una de las metas propuestas al iniciar el desarrollo de un proyecto, ya que al pasar del tiempo las herramientas que comúnmente conocemos pasan a ser obsoletas o de menor rendimiento. En este proyecto se desarrolló e implementó una red multiprotocolo en base a un sistema no comercial basado en tecnología FPGA, donde la mayor aportación del presente fue el desarrollo de software para lograr la conexión del sistema a la red Ethernet, en este punto es posible observar que al poder trabajar con hardware reconfigurable y con software que puede sujetarse a mantenimiento, se generó un sistema funcional que puede seguir en constante evolución, tanto en la parte de software como en hardware y así por lo tanto se puede adaptar a los retos futuros.

También la conexión lograda a la red Ethernet le proporciona al sistema embebido un medio de comunicación más para que este pueda compartir sus recursos desde el punto de vista como procesador y como FPGA.

Por último, el proyecto no solo demostró que la conexión a Ethernet pudo llevarse a cabo sino que además proporciona información de cómo trabajan las redes de comunicación y como se direccionan los mensajes, para así tener un punto de referencia en dado caso que se desee implementar algún otro protocolo.

5.2 PROSPECTIVAS

El desarrollo del proyecto cumplió con los objetivos propuestos a su inicio, para ello se presentaron ciertos análisis de tasas de transferencia, frecuencias de muestreo para el uso de sensores y por último la implementación de la red, todo esto demostró que se pueden generar redes que se rijan por varios medios y protocolos de comunicación, lo que permite que las redes sean flexibles para usar el protocolo que sea más conveniente en determinada circunstancia, en base a esto se observa que para agregar mayores características de flexibilidad y funcionalidad a la red de comunicaciones, es posible agregar algunos protocolos de comunicación adicionales como podrían ser RS-232 y Wi-Fi, y así generar un sistema con más opciones de comunicación.

En lo que respecta al software desarrollado para la conexión del sistema embebido a redes Ethernet, se propone una depuración del código fuente y así en complemento con el hecho de aumentar la frecuencia del módulo spi con el que se maneja la tarjeta controladora, se puedan lograr mejorar las tasas de transferencia de información bajo este protocolo.

Por último se propone que se implementen en software tal como el protocolo IP, ARP e ICMP, protocolos que cubran los niveles superiores de la arquitectura TCP/IP, con la finalidad de lograr la comunicación directa de interfaces de usuario conectadas a la red Ethernet y así dejar de prescindir de la etapa intermedia que fue la comunicación USB en este proyecto.

6.1 IMPLEMENTACIÓN DE LIBRERÍAS PARA EL CONTROL DEL ENC28J60

6.1.1 ARCHIVO DE CABECERA: ethernet.h

```
#ifndef _ETHERNET_MODULE
#define _ETHERNET_MODULE
/* Default port mapping */
#ifndef ETHERNET_BASE_ADDRESS
#define ETHERNET_BASE_ADDRESS 0x003C
#endif

#ifndef LIBRARY_CODE
/* Module ports */
port (eth_ctrl) = 0x00 +
                  ETHERNET_BASE_ADDRESS;
port (eth_stat) = 0x01 +
                  ETHERNET_BASE_ADDRESS;
port (eth_do)   = 0x02 +
                  ETHERNET_BASE_ADDRESS;
port (eth_di)   = 0x03 +
                  ETHERNET_BASE_ADDRESS;
#endif
/* Control flags */
#define ETH_CS    0x01
#define ETH_RST   0x02
#define ETH_STR   0x01
#define ETH_BUSY  0x02
/* Macro definitions */
#define eth_reset_low() eth_ctrl &= ~ETH_RST
#define eth_reset_high() eth_ctrl |= ETH_RST
#define eth_cs_low()     eth_ctrl &= ~ETH_CS
#define eth_cs_high()    eth_ctrl |= ETH_CS
#define eth_send_byte(x)   eth_do = (x); \
                       eth_stat = ETH_STR; \
                    while(eth_stat&ETH_BUSY)
#define eth_get_byte()     eth_di
/* Interrupt handlers */
#ifdef SIGNAL_ETHERNET
#ifndef SIGNAL_ETHERNET_INT
#define SIGNAL_ETHERNET_INT   0x2000
#endif
void SIGNAL_ETHERNET(void);
#endif

/* Debugging registers definition*/
#ifdef DEBUG_ETHERNET
#pragma watch eth_ctrl
#pragma watch eth_stat
#pragma watch eth_do
#pragma watch eth_di
#endif
/* Function prototypes */
// ENC28J60 macro definitions
// command codes
#define WCR (0x40)
#define BFS (0x80)
#define BFC (0xA0)
#define RCR (0x00)
#define RBM (0x3A)
#define WBM (0x7A)
#define SRC (0xFF)
// communication modes
#define HALF_DUPLEX_MODE 0
#define FULL_DUPLEX_MODE 1
// ENC28J60 control register map
#define ECON1     (0x1F)
#define ECON2     (0x1E)
#define EIR       (0x1C)
#define EREVID    (0x12)
#define ERDPTL    (0x00)
#define ERDPTH    (0x01)
#define EWRPTL    (0x02)
#define EWRPTH    (0x03)
#define ERXSTL    (0x08)
#define ERXSTH    (0x09)
#define ERXNDL    (0x0A)
#define ERXNDH    (0x0B)
#define ERXRDPTL  (0x0C)
#define ERXRDPTH  (0x0D)
#define ERXFCON   (0x18)
#define ERXWRPTL  (0x0E)
#define ERXWRPTH  (0x0F)
#define ETXSTL    (0x04)
#define ETXSTH    (0x05)
#define ETXNDL    (0x06)
#define ETXNDH    (0x07)
#define ESTAT     (0x1D)
#define EPKTCNT   (0x19)
```

```
#define MIREGADR  (0x14)
#define MICMD     (0x12)
#define MISTAT    (0x0A)
#define MIRDL     (0x18)
#define MIRDH     (0x19)
#define MIWRL     (0x16)
#define MIWRH     (0x17)
#define MACON1    (0x00)
#define MACON3    (0x02)
#define MACON4    (0x03)
#define MAIPGL    (0x06)
#define MAIPGH    (0x07)
#define MABBIPG   (0x04)
#define MAMXFLL   (0x0A)
#define MAMXFLH   (0x0B)
#define MAADR1    (0x04)
#define MAADR2    (0x05)
#define MAADR3    (0x02)
#define MAADR4    (0x03)
#define MAADR5    (0x00)
#define MAADR6    (0x01)
#define PHCON1    (0x00)
#define PHCON2    (0x10)
#define PHSTAT2   (0x11)
// sizes of the memory areas
#define RX_BUFF_START   (0x00)
// it should be odd address
#define RX_BUFF_END     (0x19F3)
#define TX_BUFF_START   (0x19F4)
// protocol flags
#define PROTOCOLO_ARP   (0x806)
#define PROTOCOLO_IP    (0x800)
#define PROTOCOLO_ICMP  (0x01)
#define PROTOCOLO_TCP   (0x06)
#define PROTOCOLO_UDP   (0x11)
//temporal identifier of protocol
#define PROTOCOLO_TMP   (0x04)
#define s_bank(x) c_reg(ECON1,3);
                  s_reg(ECON1,x)
#define reset()   eth_cs_low(); \
                 eth_send_byte(SRC);
                  eth_cs_high()
int size_buff;
int current_package;
int next_package=0;

void s_reg(char reg, char data);
void c_reg(char reg, char data);
void w_reg(char reg, char data);
void w_reg_phy(char reg, int data);
void w_buff_memory(char data);

void mw_buff_memory(char *buff,
 int length);

char r_reg_eth(char reg);
char r_reg_mac_mii(char reg);
char r_buff_memory(void);

void mr_buff_memory(char *buff,
 int length);

void send_message(void);
void freeing_buffer(void);
int r_reg_phy(char reg);
int checksum(int *data, int size);

void arp_header(char *src_IP,
 char *trgt_IP, char *MAC_ENC,
 char *MAC_target, char operation);

void mac_header(char *MAC_ENC,
 char *MAC_target, int protocol);

void ip_header(int total_length,
 int protocol_layer, char *src_IP,
 char *trgt_IP);

void icmp_header(int *data,
 int length, int ident_ICMP,
 int number_sequence);

void set_enc28j60(char mode,
 char *MAC_ENC);

void send_arp_package(char *my_MAC,
 char *my_IP, char *trgt_IP);

void send_ip_package(char *data,
 int length, char *my_MAC,
 char *my_IP, char *trgt_MAC,
 char *trgt_IP);

int receive_package(char *data,
 char *my_MAC, char *my_IP,
 char *mask_network,
 char *trgt_MAC, char *trgt_IP);
/* Link with the library implementation */
#pragma "/src/ethernet.c"
#endif
```

6.1.2 ARCHIVO FUENTE: ethernet.c

```
#include <ethernet.h>
void c_reg(char reg, char data)
{
   eth_cs_low();
   eth_send_byte(BFC|reg);
   eth_send_byte(data);
   eth_cs_high();
}
void s_reg(char reg, char data)
{
   eth_cs_low();
   eth_send_byte(BFS|reg);
   eth_send_byte(data);
   eth_cs_high();
}
void w_reg(char reg, char data)
{
   eth_cs_low();
   eth_send_byte(WCR|reg);
   eth_send_byte(data);
   eth_cs_high();
}
char r_reg_eth(char reg)
{
   char a;
   eth_cs_low();
   eth_send_byte(RCR|reg);
   eth_send_byte(0);
   a=eth_get_byte();
   eth_cs_high();
   return a;
}
char r_reg_mac_mii(char reg)
{
   char a;
   eth_cs_low();
   eth_send_byte(RCR|reg);
   eth_send_byte(0);
   eth_send_byte(0);
   a=eth_get_byte();
   eth_cs_high();
   return a;
}
void w_buff_memory(char data)
{
   eth_cs_low();
   eth_send_byte(WBM);
   eth_send_byte(data);
   eth_cs_high();
}
char r_buff_memory(void)
{
   char a;
   eth_cs_low();
   eth_send_byte(RBM);
   eth_send_byte(0);
   a=eth_get_byte();
   eth_cs_high();
   return a;    }
void mw_buff_memory(char *buff, int length)
{
   eth_cs_low();
   eth_send_byte(WBM);
   while(length--){
      eth_send_byte(*buff);
      ++buff;
   }
   eth_cs_high();
}
void mr_buff_memory(char *buff, int length)
{
   eth_cs_low();
   eth_send_byte(RBM);
   while(length--){
       eth_send_byte(0);
       *buff = eth_get_byte();
       ++buff;
   }
   eth_cs_high();
}
int r_reg_phy(char reg)
{
   s_bank(2);
   w_reg(MIREGADR, reg);
   w_reg(MICMD, 1);
   s_bank(3);
   while(r_reg_mac_mii(MISTAT)&1);
   s_bank(2);
   w_reg(MICMD, 0);
   return (r_reg_mac_mii(MIRDH)<<8) |
                   r_reg_mac_mii(MIRDL);
}
void w_reg_phy(char reg, int data)
{
   s_bank(2);
   w_reg(MIREGADR, reg);
```

```
   w_reg(MIWRL, data&0xff);
   w_reg(MIWRH, data>>8);
   s_bank(3);
   while(r_reg_mac_mii(MISTAT)&1);
}
int checksum(int *data, int size)
{
   long chks,aux;
   int i;
   chks = 0L;
   for(i=0;i<size;++i){
      aux = (unsigned int)data[i];
      chks += aux;
      if(chks&0x10000L){
         ++chks;
         chks = (unsigned int)chks;
      }
   }
   return ~chks;
}
void set_enc28j60(char mode,
 char *MAC_ENC)
{
   long a;
   eth_reset_high();
   for(a=0;a<120;a++);
   while(!(r_reg_eth(ESTAT)&1));
   reset();
   w_reg(ERXSTL, RX_BUFF_START&0xff);
   w_reg(ERXSTH, RX_BUFF_START>>8);
   w_reg(ERXNDL, RX_BUFF_END&0xff);
   w_reg(ERXNDH, RX_BUFF_END>>8);
   w_reg(ERXRDPTL, RX_BUFF_END&0xff);
   w_reg(ERXRDPTH, RX_BUFF_END>>8);
   w_reg(ETXSTL, TX_BUFF_START&0xff);
   w_reg(ETXSTH, TX_BUFF_START>>8);
   s_bank(2);
   w_reg(MAIPGL, 0x12);
   w_reg(MACON1, 0x0D);
   if(mode==1){  // full_duplex
      w_reg(MACON3, 0x31);
      w_reg(MABBIPG, 0x15);
      w_reg_phy(PHCON1, 0x0100);
   }
   else{ // half_duplex
      w_reg(MACON3, 0x30);
      w_reg(MABBIPG, 0x12);
      w_reg(MAIPGH, 0x0C);
      w_reg_phy(PHCON1, 0);
      w_reg_phy(PHCON2, 0x0100);
   }
   s_bank(2);
   w_reg(MAMXFLL, 1518&0xff);
   w_reg(MAMXFLH, 1518>>8);
   s_bank(3);
   w_reg(MAADR1, MAC_ENC[0]);
   w_reg(MAADR2, MAC_ENC[1]);
   w_reg(MAADR3, MAC_ENC[2]);
   w_reg(MAADR4, MAC_ENC[3]);
   w_reg(MAADR5, MAC_ENC[4]);
   w_reg(MAADR6, MAC_ENC[5]);
   s_bank(1);
   w_reg(ERXFCON, 0xA1);
   s_reg(ECON1, 0x04);
   a=0;
   while(a<565000L)
   a++;
}
void mac_header(char *MAC_ENC,
 char *MAC_target, int protocol)
{
   size_buff = 14;
   s_bank(0);
   w_reg(EWRPTL, TX_BUFF_START&0xff);
   w_reg(EWRPTH, TX_BUFF_START>>8);
   eth_cs_low();
   eth_send_byte(WBM);
   eth_send_byte(0);
   eth_send_byte(MAC_target[0]);
   eth_send_byte(MAC_target[1]);
   eth_send_byte(MAC_target[2]);
   eth_send_byte(MAC_target[3]);
   eth_send_byte(MAC_target[4]);
   eth_send_byte(MAC_target[5]);
   eth_send_byte(MAC_ENC[0]);
   eth_send_byte(MAC_ENC[1]);
   eth_send_byte(MAC_ENC[2]);
   eth_send_byte(MAC_ENC[3]);
   eth_send_byte(MAC_ENC[4]);
   eth_send_byte(MAC_ENC[5]);
   eth_send_byte(protocol>>8);
   eth_send_byte(protocol);
   eth_cs_high();
}
void arp_header(char *src_IP,
 char *trgt_IP, char *MAC_ENC,
 char *MAC_target, char operation)
{
   size_buff += 28;
   eth_cs_low();
   eth_send_byte(WBM);
   eth_send_byte(0);
```

```
   eth_send_byte(0x1);
   eth_send_byte(PROTOCOLO_IP>>8);
   eth_send_byte(PROTOCOLO_IP);
   eth_send_byte(0x6);
   eth_send_byte(0x4);
   eth_send_byte(0);
   eth_send_byte(operation);
   eth_send_byte(MAC_ENC[0]);
   eth_send_byte(MAC_ENC[1]);
   eth_send_byte(MAC_ENC[2]);
   eth_send_byte(MAC_ENC[3]);
   eth_send_byte(MAC_ENC[4]);
   eth_send_byte(MAC_ENC[5]);
   eth_send_byte(src_IP[0]);
   eth_send_byte(src_IP[1]);
   eth_send_byte(src_IP[2]);
   eth_send_byte(src_IP[3]);
   if(operation==2){ // reply
      eth_send_byte(MAC_target[0]);
      eth_send_byte(MAC_target[1]);
      eth_send_byte(MAC_target[2]);
      eth_send_byte(MAC_target[3]);
      eth_send_byte(MAC_target[4]);
      eth_send_byte(MAC_target[5]);
   }
   else{   // operation 1=request
      eth_send_byte(0);
      eth_send_byte(0);
      eth_send_byte(0);
      eth_send_byte(0);
      eth_send_byte(0);
      eth_send_byte(0);
   }
   eth_send_byte(trgt_IP[0]);
   eth_send_byte(trgt_IP[1]);
   eth_send_byte(trgt_IP[2]);
   eth_send_byte(trgt_IP[3]);
   eth_cs_high();
}

void ip_header(int total_length,
 int protocol_layer, char *src_IP,
 char *trgt_IP)
{
   static int ident=0;
   int suma;
   int datos[10];
   size_buff += total_length + 20;
   datos[0] = 0x4500;
   datos[1] = total_length + 20;
   datos[2] = ident;
   datos[3] = 0x4000;
   datos[4] = protocol_layer|0x8000;
   datos[5] = (src_IP[0]<<8)|src_IP[1];
   datos[6] = (src_IP[2]<<8)|src_IP[3];
   datos[7] = (trgt_IP[0]<<8)|trgt_IP[1];
   datos[8] = (trgt_IP[2]<<8)|trgt_IP[3];
   suma = checksum(datos, 9);
   eth_cs_low();
   eth_send_byte(WBM);
   eth_send_byte(0x45);
   eth_send_byte(0x00);
   eth_send_byte((total_length+20)>>8);
   eth_send_byte(total_length+20);
   eth_send_byte(ident>>8);
   eth_send_byte(ident);
   eth_send_byte(0x40);
   eth_send_byte(0x00);
   eth_send_byte(0x80);
   eth_send_byte(protocol_layer);
   eth_send_byte(suma>>8);
   eth_send_byte(suma);
   eth_send_byte(src_IP[0]);
   eth_send_byte(src_IP[1]);
   eth_send_byte(src_IP[2]);
   eth_send_byte(src_IP[3]);
   eth_send_byte(trgt_IP[0]);
   eth_send_byte(trgt_IP[1]);
   eth_send_byte(trgt_IP[2]);
   eth_send_byte(trgt_IP[3]);
   eth_cs_high();
   ident++;
}

void icmp_header(int *data,
 int length, int ident_ICMP,
 int number_sequence)
{
   int suma;
   int header_icmp[3];
   header_icmp[0] = ident_ICMP;
   header_icmp[1] = number_sequence;
   header_icmp[2] = ~checksum(data, length);
   suma = checksum(header_icmp, 3);
   eth_cs_low();
   eth_send_byte(WBM);
   eth_send_byte(0);
   eth_send_byte(0);
   eth_send_byte(suma>>8);
   eth_send_byte(suma);
   eth_send_byte(ident_ICMP>>8);
   eth_send_byte(ident_ICMP);
```

```
   eth_send_byte(number_sequence>>8);
   eth_send_byte(number_sequence);
   for(suma=0;suma<length;suma++){
      eth_send_byte(data[suma]>>8);
      eth_send_byte(data[suma]);
   }
   eth_cs_high();
}

void send_message(void)
{
   s_bank(0);
   if(r_reg_eth(EIR)&(0x2)){
       s_reg(ECON1, 0x80);
       c_reg(ECON1, 0x80);
       c_reg(EIR, 0x2);
     }
   while(r_reg_eth(ECON1)&(0x8));
   w_reg(ETXNDL,
        (TX_BUFF_START+size_buff)&(0xff));
   w_reg(ETXNDH,
        (TX_BUFF_START+size_buff)>>8);
   s_reg(ECON1, 0x8);
}

void freeing_buffer(void)
{
   int free_pointer;
   s_bank(0);
   free_pointer = next_package-1;
   if((free_pointer>RX_BUFF_END) ||
     (free_pointer<RX_BUFF_START)){
      free_pointer = RX_BUFF_END;
   }
   s_reg(ECON2, 0x40);
   w_reg(ERXRDPTL, free_pointer&(0xff));
   w_reg(ERXRDPTH,free_pointer>>8);
}

void send_arp_package(char *my_MAC,
 char *my_IP, char *trgt_IP)
{
   char aux_MAC[6]=
   {0xff,0xff,0xff,0xff,0xff,0xff};

   mac_header(my_MAC, aux_MAC,
                PROTOCOLO_ARP);
   arp_header(my_IP, trgt_IP,
           my_MAC, aux_MAC, 1);
   send_message();
}

void send_ip_package(char *data, int length,
 char *my_MAC, char *my_IP, char *trgt_MAC,
 char *trgt_IP)
{
 mac_header(my_MAC, trgt_MAC, PROTOCOLO_IP);
 ip_header(length, PROTOCOLO_TMP, my_IP,
                                    trgt_IP);
 mw_buff_memory(data, length);
 send_message();
}
int receive_package(char *data,char *my_MAC,
 char *my_IP, char *mask_network,
 char *trgt_MAC, char *trgt_IP)
{
   int a=0,rpt;
   char aux_IP[20],aux_ARP[28],mask[4];
   char control[6],header[14];
   int aux[28];

   s_bank(1);
   if(r_reg_eth(EPKTCNT)){
      current_package = next_package;
      s_bank(0);
      w_reg(ERDPTL, current_package&(0xff));
      w_reg(ERDPTH, current_package>>8);
      mr_buff_memory(control, 6);
      next_package =
                 (control[1]<<8)|control[0];
      mr_buff_memory(header, 14);
      for(rpt=0;rpt<6;rpt++)
         trgt_MAC[rpt] = header[rpt+6];
      for(rpt=0;rpt<4;rpt++)
         mask[rpt] =
               my_IP[rpt]&mask_network[rpt];
      if(((header[12]<<8)|header[13])==
                              PROTOCOLO_IP){
         mr_buff_memory(aux_IP, 20);
         if((aux_IP[16]==my_IP[0]) &&
            (aux_IP[17]==my_IP[1]) &&
            (aux_IP[18]==my_IP[2]) &&
            (aux_IP[19]==my_IP[3])){
if(((mask_network[0]&aux_IP[12])==mask[0])&&
  ((mask_network[1]&aux_IP[13])==mask[1]) &&
  ((mask_network[2]&aux_IP[14])==mask[2]) &&
   ((mask_network[3]&aux_IP[15])==mask[3])){
               for(rpt=0;rpt<10;rpt++){
                  a = rpt*2+1;
                  aux[rpt] =
               (aux_IP[rpt*2]<<8)|aux_IP[a];
               }
               a=0
```

```
            if(!checksum(aux, 10)){
               if((aux_IP[9]==PROTOCOLO_ICMP) || (aux_IP[9]==PROTOCOLO_TMP)){
                  for(rpt=0;rpt<4;rpt++)
                     trgt_IP[rpt] = aux_IP[12+rpt];
                  mr_buff_memory(data,((aux_IP[2]<<8)|aux_IP[3])-20);
                  if(aux_IP[9]==PROTOCOLO_ICMP){
                     for(rpt=0;rpt<20;rpt++){
                        a = rpt*2+1;
                        aux[rpt] = (data[rpt*2]<<8)|data[a];
                     }
                     a=0;
                     if(!checksum(aux, 20)){
                        if(data[0]==8){
                           for(rpt=4;rpt<20;rpt++){
                              a = rpt*2+1;
                              aux[rpt-4] = (data[rpt*2]<<8)|data[a]; }
                           a=0;
                           mac_header(my_MAC, trgt_MAC, PROTOCOLO_IP);
                           ip_header(40, PROTOCOLO_ICMP, my_IP, trgt_IP);
                           icmp_header(aux, 16, (data[4]<<8)|data[5],
                                                (data[6]<<8)|data[7]);
                           send_message();
                        }
                     }
                  }
                  else if(aux_IP[9]==PROTOCOLO_TMP){
                     a = ((aux_IP[2]<<8)|aux_IP[3]) - 20;
                  }
               }
            }
         }
      }
      else if(((header[12]<<8)|header[13])==PROTOCOLO_ARP){
         mr_buff_memory(aux_ARP, 28);
         if((aux_ARP[24]==my_IP[0]) && (aux_ARP[25]==my_IP[1]) && (aux_ARP[26]==my_IP[2]) &&
            (aux_ARP[27]==my_IP[3])){
            for(rpt=0;rpt<4;rpt++)
              trgt_IP[rpt] = aux_ARP[14+rpt];
              if(aux_ARP[7]==1){
                  mac_header(my_MAC, trgt_MAC, PROTOCOLO_ARP);
                  arp_header(my_IP, trgt_IP, my_MAC, trgt_MAC, 2);
                  send_message();
              }
            a=1;
         }
      }
      freeing_buffer();
   }
    return a;
}
```

6.2 CÓDIGOS FUENTE PARA IMPLEMENTACIÓN DE RED

6.2.1 CLIENTE/ETHERNET

```
// 1er tarjeta, CLIENTE ETHERNET
#define SIGNAL_USBRX on_signal_usbrx
#include <usb.h>
#include <ethernet.h>
#include <interrupt.h>
#include <timers.h>
#define ALMACENAR_IP        4
#define IDENTIFICAR         6
#define ALMACENAR_MAC       8
#define CONTROL_MONITOREO   15
#define DETENER_SISTEMA     31
#define TIME_OUT_1          12000 //  12 ms
// conectando la interrupcion
connect_signals(INT(SIGNAL_USBRX))
port led = 0x6D;
//  parametros a configurar por el usuario
char IP_E[4]={148,20,13,2};
char MAC_E[6]={0x78,0x01,0x01,
               0x02,0x03,0x06};
char mask_network[4]={255,255,255,0};
// ip del router
char IP_router[4]={148,20,13,3};
char MAC_trgt[6];
char IP_trgt[4];
char datos[50];  // minimo de 40
void main(void){
   //  Habilitando la interrupcion   por USB
   enable_int(SIGNAL_USBRX);
   // configurando el timer en microsegundos
   timer_config(0,47L);
    //  Confguraciones del ENC28J60
   set_enc28j60(HALF_DUPLEX_MODE, MAC_E);
   while(1){
      led = 1;
   }
}
void on_signal_usbrx(void)
{
   int d0, d1, d2, d3;
   static int mx, my, mz;
   static char ip0=0, ip1=0, ip2=0, ip3=0;
   static char mac0=0, mac1=0, mac2=0;
   static char mac3=0, mac4=0, mac5=0;
   char add_net[4];
   int manager_package=0;
   char u_signal;
   char sensor;
   char code;
   int tiempo=0;
   int check;
   int j;
   led = 3;
   read_usb(d0, d1, d2, d3);
   usb_rxfree();
   code = d0>>8;
   sensor = (d0&(0x0010))>>4;  // LSB
   u_signal = (d0&(0x0020))>>5;  // control
   switch(code){
      case ALMACENAR_IP:
         ip0 = d1>>8;
         ip1 = d1&(0xff);
         ip2 = d2>>8;
         ip3 = d2&(0xff);
         led = 4;
      break;
      case IDENTIFICAR:
         for(j=0;j<4;j++){//direccion de red
            add_net[j] =
                    IP_E[j]&mask_network[j];
         }
         for(j=0;j<8;j++){
       receive_package(datos, MAC_E, IP_E,
       mask_network, MAC_trgt, IP_trgt);
         }
   if(((mask_network[0]&ip0)==add_net[0]) &&
      ((mask_network[1]&ip1)==add_net[1]) &&
      ((mask_network[2]&ip2)==add_net[2]) &&
      ((mask_network[3]&ip3)==add_net[3])){
            IP_trgt[0]=ip0;
            IP_trgt[1]=ip1;
            IP_trgt[2]=ip2;
            IP_trgt[3]=ip3;
         }
         else{// se asigna ip de router
            for(j=0;j<4;j++){
               IP_trgt[j] = IP_router[j];
            }
         }
         send_arp_package(MAC_E, IP_E,
                  IP_trgt); // se pide MAC
         timer_start(0);
         check=1;
```

```
    while(check){
       led=4;
       manager_package =
    receive_package(datos, MAC_E, IP_E,
      mask_network, MAC_trgt, IP_trgt);
       tiempo=timer_get_count(0);
       if(manager_package==1){
          check=0;
          d0 = (MAC_trgt[0]<<8) |
                      (MAC_trgt[1]);
          d1 = (MAC_trgt[2]<<8) |
                      (MAC_trgt[3]);
          d2 = (MAC_trgt[4]<<8) |
                      (MAC_trgt[5]);
          d3 = 15;
       }
       if(tiempo>TIME_OUT_1){
          check=0;
          d0 = 0;
          d1 = 0;
          d2 = 0;
          d3 = 0;
       }
    }
    timer_stop(0);
    timer_clear_count(0);
    write_usb(d0,d1,d2,d3);
    usb_txsend();
    led = 5;
 break;
 case ALMACENAR_MAC:
    mac0 = d1>>8;
    mac1 = d1&(0xff);
    mac2 = d2>>8;
    mac3 = d2&(0xff);
    mac4 = d3>>8;
    mac5 = d3&(0xff);
    led = 6;
 break;
 case CONTROL_MONITOREO:
    if(sensor==0){ // accel
       datos[0] = 4;  // adress slave
       datos[1] = CONTROL_MONITOREO;
       datos[2] = u_signal;
       IP_trgt[0] = ip0;
       IP_trgt[1] = ip1;
       IP_trgt[2] = ip2;
       IP_trgt[3] = ip3;
       MAC_trgt[0] = mac0;
       MAC_trgt[1] = mac1;
       MAC_trgt[2] = mac2;
       MAC_trgt[3] = mac3;
       MAC_trgt[4] = mac4;
       MAC_trgt[5] = mac5;
       send_ip_package(datos, 3, MAC_E,
              IP_E, MAC_trgt, IP_trgt);
       timer_start(0);
       check=1;
       while(check){
          led=6;
          manager_package =
    receive_package(datos, MAC_E, IP_E,
      mask_network, MAC_trgt, IP_trgt);
          tiempo=timer_get_count(0);
          if(manager_package>1){
             check=0;
             d3 = datos[12];
          }
          if(tiempo>TIME_OUT_1){
             check=0;
             d3 = 6;
          }
       }
       timer_stop(0);
       timer_clear_count(0);
       d0 = (datos[1]<<8)|datos[0];
       d1 = (datos[3]<<8)|datos[2];
       d2 = (datos[5]<<8)|datos[4];
       mx = (datos[7]<<8)|datos[6];
       my = (datos[9]<<8)|datos[8];
       mz = (datos[11]<<8)|datos[10];
       led=7;
       write_usb(d0,d1,d2,d3);
       usb_txsend();
    }
    else if(sensor==1){
       led=8;
       write_usb(mx, my, mz, d3);
       usb_txsend();
    }
 break;
 case DETENER_SISTEMA:
    datos[0] = 4;
    datos[1] = DETENER_SISTEMA;
    datos[2] = 0;
    send_ip_package(datos, 3, MAC_E,
              IP_E, MAC_trgt, IP_trgt);
    led=10;
 break;
 }
}
```

6.2.2 SERVIDOR/ETHERNET - MAESTRO/MODBUS

```
// 2da, Servidor Ethernet y Master RS-485
#define SIGNAL_TIMEOUT0 modbus_timeout
#include <ethernet.h>
#include <RS485.h>
#include <timers.h>
#include <interrupt.h>
connect_signals(INT(SIGNAL_TIMEOUT0))
port led = 0x6D;
//  parametros a configurar por el usuario
char MAC_E[6]={0x78,0x04,0x10,0x00,0x00,
                      0x10}; //direccion MAC
char IP_E[4]={148,20,13,4};  // direccion IP
char mask_network[4]={255,255,255,0};
char MAC_trgt[6];
char IP_trgt[4];
char datos[50];//debe ser de un minimo de 40
int flg;
void main(void){
   int i=0;
   char function;
   // interrupcion de 6 ms
   timer_config(0, 288000L);
   enable_int(SIGNAL_TIMEOUT0);
   //  Configuraciones de servidor Ethenet
   set_enc28j60(HALF_DUPLEX_MODE, MAC_E);
   //  Configuraciones de maestro RS-485
   rs485_bauds(RS485_DIV_BY_1); // 800KHz
   led = 1;
    while(1){
      //  recibiendo informacion del cliente
      i=receive_package(datos, MAC_E, IP_E,
           mask_network, MAC_trgt, IP_trgt);
      led=2;
       if(i>1){
         led=3;
         function=datos[1];
         if(function==15){
            rs485_push(datos[0]);
            rs485_push(datos[1]);
            rs485_push(datos[2]);
            rs485_send();
            rs485_wait_tx();
            rs485_flush();
            led=4;
            timer_start(0);
            flg=1;
          while((!rs485_received())&&(flg));
            timer_stop(0);
            timer_clear_count(0);
            led=5;
            if(!flg){
               datos[12] = 8;
            }
            else{
               rs485_pop(datos[0]);
// comienza el acelerometro por el LSB
               rs485_pop(datos[1]);
               rs485_pop(datos[2]);
               rs485_pop(datos[3]);
               rs485_pop(datos[4]);
               rs485_pop(datos[5]);
               rs485_pop(datos[6]);
// comienza el magnetometro
               rs485_pop(datos[7]);
               rs485_pop(datos[8]);
               rs485_pop(datos[9]);
               rs485_pop(datos[10]);
               rs485_pop(datos[11]);
               rs485_pop(datos[12]);
//  lleva el valor de un posible timeout
               rs485_flush();
            }
// enviando informacion al cliente Ethernet
            send_ip_package(datos, 13,
            MAC_E, IP_E, MAC_trgt, IP_trgt);
            led = 15;
         }
         else if(function==31){
            led = 8;
            rs485_push(datos[0]);
            rs485_push(datos[1]);
            rs485_push(datos[2]);
            rs485_send();
            rs485_wait_tx();
            rs485_flush();
         }
      }
   }
}

void modbus_timeout()
{
   flg = 0;
}
```

6.2.3 ESCLAVO/MODBUS - MAESTRO/ENHANCED SHOCKBURST

```
// 3er tarjeta, Slave 485 y Master RF
#define SIGNAL_TIMEOUT0 rf_timeout
#include <timers.h>
#include <RS485.h>
#include <nrf2401.h>
#include <das.h>
#include <interrupt.h>
#define DATA_SIZE 15
connect_signals(INT(SIGNAL_TIMEOUT0))
port led = 0x6D;
int flg;
void main(void)
{
   int pipe;
   char msg[DATA_SIZE];
   char address, fun, signal;
   // interrupcion de 6 ms
   timer_config(0, 192000L);
   enable_int(SIGNAL_TIMEOUT0);
    //  Configuraciones de transceptor RF
   nrf_set_tx_addr(0x78787878L);
   nrf_set_rx0_addr(0x78787878L, DATA_SIZE);
   nrf_init(4, NRF_ERX_P0);
   rs485_bauds(RS485_DIV_BY_1); // 800KHz
   while(1){
      led = 3;
//Recibiendo informacion del maestro RS-485
      while(!rs485_received());
      rs485_pop(address);
      rs485_pop(fun);
      rs485_pop(signal);
      rs485_flush();
      led = 6;
        if(address==4){
         if(fun==15){
            msg[0] = 12;
            nrf_send_data(msg, DATA_SIZE);
            outputs(signal);
            timer_start(0);
            flg=1;
            led=13;
            delay_us(1,900);
           while((!nrf_check_int())&&(flg));
            timer_stop(0);
            timer_clear_count(0);
            if(flg){
               pipe = nrf_get_rx_pipe();
               nrf_get_data(msg, pipe);
               led=14;
               rs485_push(msg[1]);
               rs485_push(msg[2]);
               rs485_push(msg[3]);
               rs485_push(msg[4]);
               rs485_push(msg[5]);
               rs485_push(msg[6]);
               rs485_push(msg[7]);
               rs485_push(msg[8]);
               rs485_push(msg[9]);
               rs485_push(msg[10]);
               rs485_push(msg[11]);
               rs485_push(msg[12]);
               rs485_push(msg[13]);
               rs485_send();
               rs485_wait_tx();
               rs485_flush();
               led=15;
            }
            else{
               rs485_push(1);
               rs485_push(2);
               rs485_push(3);
               rs485_push(4);
               rs485_push(5);
               rs485_push(6);
               rs485_push(7);
               rs485_push(8);
               rs485_push(9);
               rs485_push(10);
               rs485_push(11);
               rs485_push(12);
               rs485_push(10);
               rs485_send();
               rs485_wait_tx();
               rs485_flush();
               led=15;
            }
         }
         else if(fun==31){
            outputs(0);
            led = 8;
         }
      }
   }
}
void rf_timeout()
{
   flg = 0;
}
```

6.2.4 ESCLAVO/ENHANCED SHOCKBURST

```
// 4ta tarjeta, funciona como esclavo Rf
#include <nrf2401.h>
#include <imu.h>
#define DATA_SIZE 15  // debe ser igual al maestro
port led = 0x6D;

void main(void)
{
   int pipe, i;
   int x, y, z, mx, my, mz;
   char msg[DATA_SIZE];

    // configuraciones de transceptor RF
   nrf_set_tx_addr(0x78787878L);
   nrf_set_rx0_addr(0x78787878L, DATA_SIZE);
   nrf_init(4, NRF_ERX_P0);

   while(1){
      led = 2;
      //  Recibiendo informacion del maestro RF
      if(nrf_check_int()){ // Check for activity
         pipe = nrf_get_rx_pipe(); // Get active pipe
         nrf_get_data(msg, pipe); // Get received data on pipe

         led = 10;
         sync_read_accel(x, y, z);
         sync_read_mag(mx, my, mz);

         msg[1] = x;
         msg[2] = x>>8;
         msg[3] = y;
         msg[4] = y>>8;
         msg[5] = z;
         msg[6] = z>>8;

         msg[7] = mx;
         msg[8] = mx>>8;
         msg[9] = my;
         msg[10] = my>>8;
         msg[11] = mz;
         msg[12] = mz>>8;
         msg[13] = 12;   //  no hay problema en la comunicacion

         //  Enviando la informacion al maestro RF
         nrf_send_data(msg, DATA_SIZE);
         led = 15;
      }
   }
}
```

7. REFERENCIAS

[1] Axelson, Jan. Serial Port Complete Programming and Circuits for RS-232 and RS-485 Links and Networks, 3rd ed.; Lakeview research, USA, 1998.

[2] Cura Norberto Julián. Transformación de direcciones IP en direcciones físicas (ARP – Protocolo de resolución de direcciones), 2010.

[3] Candelas Herías, Francisco Andrés. Comunicación con RS-485 y MODBUS. Grupo de Innovación Educativa en Automática, Universidad de Alicante, 2011.

[4] Curiel Razo, Yajaira Ilse. Monitoreo del comportamiento térmico de distintos recubrimientos en techos mediante el uso de una red inalámbrica de sensores (WSN) para la predicción de temperatura con redes neuronales, Tesis de maestría, Universidad Autónoma de Querétaro, México, 2011.

[5] Durán Marroquín, Claudio Iván, Gutiérrez Rojas, Emmanuel. Diseño y aplicación de un sistema de adquisición y transmisión de datos vía ethernet, Tesis de Ingeniería, Universidad Autónoma del Estado de Hidalgo, México, 2009.

[6] Duarte Correa, David. Implementación de un control difuso inalámbrico de un sistema experto en granjas acuícolas, Tesis de maestría, Universidad Autónoma de Querétaro, México, 2009.

[7] González Cruz, Claudia Aide. Diseño de una red de sensores para control de iluminación basado en lógica difusa, Tesis de maestría, Universidad Autónoma de Querétaro, México, 2012.

[8] Gordon Pico, Mario Emiliano, Vásquez Castro, Bryan Sebastián. Análisis de desempeño de redes de sensores inalámbricas en tiempo real aplicadas a monitorización volcánica, Proyecto de grado, Escuela Politécnica del Ejército, Ecuador, 2012.

[9] Guzmán Gallegos, Ricardo, Velasco Mendoza, Rodrigo. Monitoreo y control de riego mediante una red Ethernet con interfaz en LabView. Adquisición de datos en una red Modbus Plus a través de una red Interbus, Tesina de licenciatura, Universidad Autónoma de Querétaro, México, 2009.

[10] Frank, Randy. Understanding Smart Sensors. 2nd ed.; Artech House, USA, 2000.

[11] https://developer.gnome.org/gtkmm-tutorial/stable/index.html.

[12] Iberico Costa, Roberto. Diseño de un sistema de seguridad basado en una red actuador-sensor zigbee con soporte en la WLAN de un edificio de departamentos, Tesis de ingeniería, Pontificia Universidad Católica del Perú, Perú, 2010.

[13] Lugo Álvarez, Erick Manuel. Desarrollo de protocolo Ethernet en FPGA para el procesamiento digital de datos de cámara termográfica, Tesis de maestría, Universidad Autónoma de Querétaro, México, 2013.

[14] Márquez, J.; Pardo, K.; Pizarro, S. Ethernet: Su origen, funcionamiento y rendimiento. Ingeniería y Desarrollo. Universidad del Norte 2001, 9, 22-34.

[15] Mayné, Jordi. Estado actual de las comunicaciones por radio frecuencia, SILICA Avnet Iberia, Rev. 4, 2009.

[16] Mejía Barrón, Arturo. Monitoreo vía Ethernet de la calidad de la energía en sistemas de baja tensión, Tesis de licenciatura, Universidad Autónoma de Querétaro, México, 2013.

[17] Microchip, 2006-2012, ENC28J60 Stand-Alone Ethernet Controller with SPI Interface.

[18] Microchip, 2010, ENC28J60 Silicon Errata and Data Sheet Clarification.

[19] Módulo Ethernet ENC28J60 para microcontroladores PIC. Manuscrito anónimo y no publicado. Recuperado de:

http://www.ucontrol.com.ar/forosmf/explicaciones-y-consultas-tecnicas/ethernet-por-donde-empiezo/?action=dlattach;attach=4389.

[20] Morales Velázquez, Luis. Sistemas Embebidos (Apuntes del Curso). 1ra ed.; Universidad Autónoma de Querétaro, México, 2013.

[21] Nordic Semiconductor, 2008, nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification V1.0.

[22] Osio, J.; Antonini, L.; Aróztegui W.; Rapallini J. Descripción General de un Microcontrolador (Módulos de Comunicación), Universidad Nacional de la Plata, Facultad de ingeniería, 2011.

[23] Rivera, J.; Herrera, G.; Chacón, M.; Acosta, P.; Carrillo, M. Improved progressive polynomial algorithm for self-adjustment and optimal response in intelligent sensors. Sensors 2008, 8, 7410-7427.

[24] Rodríguez Penin, Antonio. Sistemas SCADA. 2da ed.; MARCOMBO, España, 2007.

[25] Stallings, William. Comunicaciones y Redes de Computadores. 6ta ed.; Pearson Educación, España, 2000.

[26] Toledano Ayala, Manuel. Diseño e implementación de un sistema de monitoreo remoto para un invernadero, Tesis de maestría, Universidad Autónoma de Querétaro, México, 2006.

[27] National Instruments. Introducción a la Tecnología FPGA: Los Cinco Beneficios Principales, 2011. Recuperado de: http://www.ni.com/white-paper/6984/es/.

[28] Vera Salas, Luis Alberto. Sistema de medición y análisis de vibraciones inalámbrico basado en FPGA, aplicado a una máquina ensambladora de componentes de chip, Tesis de maestría, Universidad Autónoma de Querétaro, México, 2010.